Figure 1: rusty

Yanlin Duan (yd2380) - System Architect

Zhuo Kong (zk2202) - Tester

Emily Meng (ewm2136) - Language Guru

Shiyu Qiu (sq2156) - Manager

Summary

Overview

Introduction

Motivation

C is a powerful and widely used language. However, sometimes C can be tricky - we've all had our share of headaches with segmentation faults and data races. When deciding the fundamental goal of our project, we knew that we wanted our language to be easy to use, but still flexible with a high degree of control over the computer. Additionally, we wanted our language to be memory-safe. Thus, we were inspired by Rust, a systems programming language that prevents segmentation faults and manages data concurrency. We hope to build our own language that captures the essence of Rust, a "rusty" version of Rust, if you will. We present rusty, a safer systems programming language that combines features from Rust with some of the conveniences of higher level programming languages.

Description

Rusty aims to be a version of the Rust programming language, keeping its key features of safety in memory but shedding some of the other extraneous features to try and retain the speed of C as much as possible. It can be used as a general purpose programming language with a focus on helping the programmer avoid many of the common memory safety-related pitfalls when programming in C, such as segmentation faults or null pointers, as well as providing useful features often found in higher level programming languages.

Rusty will allow only be one binding per variable, so when memory is moved to a new variable, the old variable is no longer associated with that memory. Since passing memory back and forth between variables is a pain, Rusty will use a Borrow feature implemented with references to variables, essentially borrowing ownership of that memory and will not deallocate after going out of scope. There will be two types of references, denoted by & and &mut. The former indicates the borrowed memory is immutable, while the latter allows a user to change the memory. For any given variable, only one of the two types of reference will be allowed at any time. There can be multiple & references or only one &mut reference to keep concurrency and avoid data races.

C is very powerful, but with great power there is great responsibility - Rusty will help programmers manage some of the intricacies and care needed when using C, so they can spend more effort on the design of their programs rather than worry about accidentally dereferencing a null pointer or accessing invalid memory. The days of segmentation faults are over. Target users include systems programmers and people who have suffered enough while learning the semantics of C and can fully appreciate the features of Rusty.

Key Features

- Memory safety: avoid data races and segfaults with the ownership system and borrow checker
- Zero-cost abstraction: all memory safety checks performed at compile-time, no run-time overhead incurred
- Scoping rules: support variable shadowing
- Aggregated data structures: arrays and structs, heap allocation for more freedom in regards to referencing/lifetime
- Statically and strongly-typed: catch easy to miss bugs at compile-time
- Cross-platform: compile once and run everywhere with LLVM IR

Language Tutorial

Environment Setup

The rusty compiler has been tested on a Ubuntu 16.04 virtual machine running in VirtualBox.

There are various packages that need to be installed before the compiler can be compiled. The following commands are used to install LLVM and its development libraries:

```
sudo apt-get install -y ocaml m4 llvm opam
opam init
opam install llvm.3.8 ocamlfind
eval `opam config env`
```

Next, download the rusty compiler. The following command is used to clone the repository containing the rusty compiler:

```
git clone git@bitbucket.org:sherryq/rusty.git
```

Running the Compiler

Inside the 'rusty' directory, the following command is used to compile the rusty compiler:

```
make
```

The compiler will compile '.rusty' files into '.ll' files, which are LLVM IR files. Run the rusty executable to create a '.ll' file and run the '.ll' file with the following commands:

```
./rusty.native < test.rusty > test.ll
lli test.ll
```

Function Syntax

Functions are declared with the keyword 'fn' followed by its name, an optional parameter list in parentheses, the output arrow, and its return type. The body of a function is enclosed in brackets, defined by a sequence of statements.

A simple Hello World function:

```
fn helloWorld() -> int {

    println("hello world!");

}
```

Primitives

rusty supports the following primitives: int, float, bool, char.

Primitives are declared with the keyword 'let' followed by an identifier, a colon, the primitive type (int, float, bool, char), an equals sign, and the value that will be bound to the identifier.

Examples of primitive type declarations:

```
let x: int = 42;
let f: float = 42.0;
let b: bool = true;
let c: char = 'c';
```

Variable Binding

In rusty, there is a difference between variable binding and assignment.

An example of variable binding, which binds the resource, an integer 42, to the identifier x:

```
let x: int = 42;
```

It is possible to re-bind the identifier x anytime, even with different type, but it is not possible to re-assign x in the following usage:

```
let x: int = 42;
x = 43; /* Error: change an immutable value! */

let x: string = "hello world"; /* this is fine */
println(x);                    /* hello world */
```

To allow re-assignment, the variable must be declared as mutable with the keyword 'mut':

```
let mut x: int = 42;
x = 43;             /* this time it is valid */
println(x);         /* 43 */
```

Scoping

rusty supports variable shadowing, where the same identifier can be used to bind variables in different scopes.

```
let x: int = 42;
{
    let x: int = 43;
    println(x) /* 43 */
}
println(x); /* 42 */
```

To learn more about the rules related to variable binding and ownership, read the Memory Safety section.

Data Structures

Array

rusty supports array creation of a specified size and array access. Arrays contain elements of the same type, and can be created and accessed as follows:

```
let x:[int;5] = [1,2,3,4,5];
println(x[0]); /* prints 1 */

let y:[string;7] = ["mon", "tues", "wed", "thurs", "fri", "sat", "sun"];
let i:int = 0;
for (i = 0; i<7; i= i+1) {
    println(y[i]); /* prints each day of the week */
}
```

Struct

rusty supports custom data types through struct, which is a single data type that combines variables with identifiers as field labels. structs can have associated methods, which are defined with the keyword 'impl' followed by its struct type. A struct must be defined before any associated methods can be defined, and only variables of the declared struct type can use the associated method:

```
/* struct definition */
struct Point {
        x: int,
        y: int
}

/* struct creation */
let origin : Point = {x:10,y:20};
/* struct method definition */
impl Point{
        fn test() -> int{
                println("magic");
                return 42;
        }
}

/* struct method call */
let x: int = origin::test();
```

Operators

Binary Operators

rusty supports the following binary operators:

Arithmetic (+ , - , * , / , %)

Logical (and, or)

Relational (== , != , <, >, <=, >=)

Unary Operators

rusty supports the following unary operators:

Arithmetic (-)

Logical (not)

Memory (&, &mut, *)

Binary and unary operators are intuitive to use.

```
fn arithmetic(x:int, y:int) -> int {
    println(x+y);
    println(x-y);
    let z:int = x%y;

    if (true or false) {
        println("is this true or false?");
    }
    if (not true) {
        println("this will not ever print.");
    }

    return 0;
}
```

There are several special unary operators that are related to rusty's memory safety functionalities.

To borrow a resource from a variable instead of taking ownership:

```
let x: int = 42;
let y: &int = &x; /* y borrows x */
println(*y);      /* dereference y and print, see 42 in the console */
println(x);       /* you may still access x after borrowing, see 42 in the
    console */
```

Similarly, rusty supports mutable borrowing:

```
let mut x: int = 42;
let mut y: &mut int = &mut x; /* y mutably borrows x. */
*y = *y + 1;
println(*y);                  /* dereference y and print, see 43 in the
    console */
println(x);                   /* errors out, try to immutable borrow x on
    this line, which is not allowed. */
```

As you can see from line 2, we deliberately make mutable borrow verbose since you are granting a borrow for not only read access, but also write access, so you should be cautious.

Control Flow

rusty supports if-then/else-then conditional branching and for/while loops with C-style syntax.

A program is allowed to execute certain lines of code based on a condition with if-then/else-then conditional branching. A program is allowed to continuously execute the same lines of code as long as a condition is true with while looping. A program is able to iterate over a changing variable with for looping.

Examples:

```
let a: int = 3;
let b: int = 4;
while (a != b) {
    if (a > b) {
        a = a-b;
    }
    else {
        b = b-a;
    }
}

let i: int = 1;
for (i = 0; i < 12; i = i+1) {
    println(i);
}
```

Language Reference Manual

1. Introduction

rusty is a condensed version of the Rust programming language, based on C. rusty retains Rust's key feature of memory safety. rusty can be used as a general purpose programming language, but is intended to be a safer systems programming language. rusty uses the LLVM as a compiler IR, aligning with the goal of keeping rusty as a fast, but flexible language.

2. Lexical Conventions

2a. Tokens

There are five classes of tokens: identifiers, keywords, literals, operators, and separators.

White space such as blanks, tabs, and newlines are ignored. However, there are cases where they are required to separate tokens.

```
let whitespace = [' ' '\t' '\r']
let newline = '\n'
```

Comments, denoted as beginning with the characters /* and terminating with the characters */, are also ignored. They do not nest and cannot appear within literals.

```
and comment = parse
    newline { incr lineno; comment lexbuf }
```

```
  |    "*/"      { decr depth; if !depth > 0 then comment lexbuf else token
     lexbuf }
  | "/*"       { incr depth; comment lexbuf }
  |    _        { comment lexbuf }

and comment2 = parse
    newline {token lexbuf}
  | _ {comment lexbuf}
```

2b. Identifiers

Identifiers are sequences of mainly letters and digits, but usage of the underscore is allowed. Upper and lowercase letters are considered to be distinct. The first character of an identifier cannot be a number. An identifier cannot be the same as an existing keyword.

```
let alpha = ['a'-'z' 'A'-'Z']
let digit = ['0'-'9']
let id = alpha (alpha | digit | '_')*
```

2c. Keywords

Keywords are specific identifiers reserved for use by the language and may not be used otherwise.

```
bool
char
else
false
float
for
if
int
let
in
mut
return
string
struct
true
void
while
and
or
not
loop
fn
impl (struct methods)
```

2d. Literals

Literals are sequences of characters, whose valid character sets vary depending on type of use.
The literals are:

- Integer
- Float
- Boolean
- Character
- String

Integer literals are sequences of digits representing only whole number values in decimal.

```
let digit = ['0'-'9']
let int = digit+
```

Float literals include an integer part, a decimal point, a fraction part, an optional e or E with integer exponent. The integer part, fraction part, and integer exponent are both sequences of digits. Either the integer or the fraction part may be missing, but not both.

```
let float = (digit+) ['.'] digit+
```

Boolean literals are one of two valid sequences of characters, true or false.

```
"true"   { TRUE }
| "false"  { FALSE }
```

Character literals are single character sequences surrounded by single quotes. Special characters are represented with an escape sequence of a single backslash and a character.

'\'' - single quote

'\"' - double quote

```
let ascii = ([' '-'!' '#'-'[' ']'-'~'])
let char = ''' ( ascii | digit ) '''
let escape = '\\' ['\\' ''' '"' 'n' 'r' 't']
let escape_char = ''' (escape) '''
```

String literals are sequences of characters surrounded by double quotes. The valid character set includes anything that can be represented by a character literal.

```
let string = '"' ( (ascii | escape)* as s) '"'
```

2e. Operators

Operators are specific lexical elements reserved for use by the language and may not be used otherwise. Refer to the Expressions section for functionality and use cases.

- Arithmetic: + - * / %
- Assignment: =
- Equivalence: == != < <= > >=
- Logical: and, or, not
- Reference: & mut &mut *

2f. Separators

Separators are specific lexical elements reserved for use by the language and may not be used otherwise. They are responsible for denoting the separation between tokens. White space is considered a separator.

```
( ) { } [ ] ; : , .
```

3. Types

3a. Primitive Data Types

int

The integer type stores whole number values.

```
let x : int =32;
```

float

The float type stores fractional number values.

```
let y: float = 0.4;
```

bool

The bool type represents either true or false.

```
let y: bool = false;
```

void

The void type represents an empty value used only as the type returned by functions that do not generate values. It cannot be used as a variable type.

```
fn print_num(x: int)-> void {
  println“”({}, x);
}
```

char

The char type represents a single character surrounded by single quotes stored in 8 bits.

```
let x: char ='x';
```

string

The string type represents a sequence of characters either as a literal constant or as some kind of variable.

```
let hello: string = "Hello, world!";
```

3b. Non-Primitive Data Types

array

The array type represents a fixed-size array, denoted let <arrayname>: [type; size] = [elements] in definition.

Syntax:

```
let array1: [int;4] = [1,2,3,4];
println(array1[0]);
```

Grammar:

```
LBRACK typ SEMI INT_LITERAL RBRACK {ArrayT($2,$4)} /* array type */
LBRACK expr_list RBRACK { ArrayLit(List.rev $2) } /* array  creation */
expr LBRACK expr RBRACK { ArrayAccess($1,$3) } /* array access */
```

struct

The struct type is a single, unified data type that combines variables with names of these variables as field labels.

Syntax:

```
struct Point {
    x: int,
    y: int,
}

fn main()-> void {
    let origin: Point = { x: 0, y: 0 };
    println(origin.x);
}
```

Grammar:

```
ID { StructT($1) } /* struct type */
STRUCT ID LBRACE formals_opt RBRACE { StructDef($2,$4) } /* struct definition
    */
LBRACE struct_list_opt RBRACE { StructCreate($2) } /* struct creation */
ID COLON COLON ID LPAREN actuals_opt RPAREN { StructAccess($1,$4,$6) } /*
    access struct */
```

3c. Type Qualifiers

mut

The mut qualifier indicates that the value of the variable is mutable, meaning its value and memory can be borrowed by another variable.

Syntax:

```
let mut x: int = 5;
x = 6; // mutable x allows changing 'xs contents
let y: &mut int = &mut x; // y is referencing/borrowing 'xs resource and can
    change 'xs contents
```

Grammar:

```
MUTABLEBORROW typ { RefT(Mut,$2) }
MUTABLEBORROW expr { Unop(Borrow(Mut), $2) }
LET MUTABLE ID COLON typ ASSIGN expr SEMI { Declaration(Mut,($3,$5),$7) }
```

4. Operators

4a. Arithmetic

The binary arithmetic operators are +, −, *, /, %.

Integer division truncates any fractional part.

The binary + and − operators have the same precedence, which is lower than the precedence of *, /, %. Arithmetic operators associate left to right.

Syntax:

```
let x:int = (2+3) * 4;
```

Grammar:

```
expr:
  expr PLUS   expr { Binop($1, Add,   $3) }
    | expr MINUS  expr { Binop($1, Sub,   $3) }
    | expr TIMES  expr { Binop($1, Mult,  $3) }
    | expr DIVIDE expr { Binop($1, Div,   $3) }
    | expr MODULO expr { Binop($1, Mod,   $3) }
```

4b. Assignment

The operator = assigns an expression to a variable. Assignment operator is right associative.

Syntax:

```
let x:int = 3;
```

Grammar:

```
LET ID COLON typ ASSIGN expr SEMI  { Assign(Immut, ($4, $2), $6) }
| LET MUTABLE ID COLON typ ASSIGN expr SEMI { Assign(Mut, ($5,$3),$7) }
```

4c. Equivalence

The relational operators are <, <=, >, >=. They all have the same precedence. Below them are == and !=.
Relational operators have lower precedence than arithmetic operators.

Syntax:

```
let x:bool = (3 == 3);
```

Grammar:

```
expr:
  expr EQ     expr { Binop($1, Equal, $3) }
    | expr NEQ    expr { Binop($1, Neq,   $3) }
    | expr LT     expr { Binop($1, Less,  $3) }
    | expr LEQ    expr { Binop($1, Leq,   $3) }
    | expr GT     expr { Binop($1, Greater, $3) }
    | expr GEQ    expr { Binop($1, Geq,   $3) }
```

4d. Logical

The logical operators are and, or, not.

Syntax:

```
if (not valid)
if (valid or not_valid)
if (a_valid and b_valid)
```

Grammar:

```
expr:
  expr AND    expr { Binop($1, And,    $3) }
  | expr OR     expr { Binop($1, Or,     $3) }
  | NOT expr         { Unop(Not, $2) }
```

4e. Memory

The unary reference operator & applied to a variable borrows ownership of the variable's resource but continues to be immutable.

The unary mutable reference operator &mut applied to a variable's lvalue allows it's rvalue to borrow ownership of the resource and allows the borrowed resource to be mutated.

The unary dereference operator * applied to a mutable reference variable will dereference it and give access to the resource or actual contents of a reference.

Grammar:

```
TIMES expr %prec DEREF { Unop(Deref, $2) } /* * */
BORROW expr {Unop(Borrow(Immut), $2)} /* borrow */
MUTABLEBORROW expr {Unop(Borrow(Mut), $2)} /* mut borrow */
```

4f. Other unary operator expressions

Unary operator − gives negative signs to numeric values. It has higher precedence than arithmetic operators.

Syntax:

```
let x:int = -3;
```

Grammar:

```
MINUS expr %prec NEG { Unop(Neg, $2) } /* - */
```

5. Statements and Expressions

5a. Declarations and lvalues/rvalues

Variables are declared through let statements that are followed by a type annotation and an initializer expression. Any variables created through definitions are valid from the point of definition until the end of its enclosing scope.

Syntax:

```
let x: int = 5;
```

Expressions are divided into lvalues and rvalues. Within each expression, subexpressions occur in either lvalue context or rvalue context. Evaluation of an expression relies on whether it is an lvalue or rvalue and the context in which the expression occurs.

Lvalues are expressions representing memory locations. All other expressions are rvalues.

When an lvalue is evaluated in an lvalue context, it denotes a memory location. When an lvalue is evaluated in an rvalue context, it denotes the value held in that memory location.

5b. Literal expressions

A literal expression consists of one of the literal forms described earlier or the unit value, and ending with a semicolon.

Syntax:

```
"rusty";    // string literal expression
'r';        // character literal expression
42;         // integer literal expression
42.35;      // float literal expression
```

5c. Array expressions

An array expression is written by enclosing zero or more comma-separated expressions of uniform type in square brackets.

Syntax:

```
[1, 2, 3, 4]  // array of integers“
["a, “"b, “"c, “"d, “"e] // array of characters
```

5d. Struct expressions

A struct expression is written as a brace-enclosed list of zero or more comma-separated name-value pairs, providing the field values of a new instance of the struct.

For example, given a struct definition:

```
struct Point { x: float, y: float }
```

We can create a new Point struct like this:

```
let f: Point = {x: 10.0, y: 20.0};
```

We may also associate methods to struct using keyword impl and call it using syntax below:

```
impl Point: {

  fn sumXY(&self: &struct) -> float {
    return (self.x + self.y);
}

let f: Point = {x: 10.0, y: 20.0};
println(f::sumXY());
```

5f. Function calls

A function item defines a sequence of statements and a final expression, along with a name and a set of parameters. Other than a name, all these are optional. Functions are declared with the keyword fn.

Functions may declare a set of input variables as parameters, through which the caller passes arguments into the function, and the output type of the value the function will return to its caller on completion.

A hello-world function will look like this:

```
fn helloWorld() -> void {
    println"(Hello world"!)
}
```

Grammar

```
fdecl: /* fn foo(x:int) -> int { ... } */
    FUNC ID LPAREN formals_opt RPAREN OUTPUT typ LBRACE stmt_list RBRACE
     {
         fname = $2;
         formals = $4;
         outputType = $7;
         body = List.rev $9 }

ID LPAREN actuals_opt RPAREN { Call($1, $3) } /* Function Call */
```

5g. Control flow

if/else

An if expression is a conditional branch in program control. The form of an if expression is a condition expression, followed by a consequent block, and an optional trailing else block.

```
if (expr) {
 (block)
}…

else {
 (block)
}
```

The condition expressions must have type bool. If a condition expression evaluates to true, the consequent block is executed and any subsequent else if or else block is skipped. If a condition expression evaluates to false, the consequent block is skipped and any subsequent else if condition is evaluated. If all if and else if conditions evaluate to false then else block is executed (if any).

Grammar:

```
    IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt     { If($3, $5, $7) }
```

loops

rusty supports three kinds of loops: for, while, and loop (infinite loop).

- for

A for expression is used for iterating a specified number of times, executing the statements inside of the loop each time.

Usage of for loops looks like this:

```
let i:int = 0;

for(i = 0; i < 5; i = i+1) {
   println(i); /* prints 1, 2, 3, 4 */
}
```

Grammar:

```
FOR LPAREN expr SEMI expr SEMI expr RPAREN stmt { For($3, $5, $7, $9) }
```

- while

A while loop begins by evaluating the boolean loop conditional expression. If the loop conditional expression evaluates to true, the loop body block executes and control returns to the loop conditional expression. If the loop conditional expression evaluates to false, the while expression completes.

Syntax:

```
while (expr) {
 (block)
}

let mut i:int = 0;

while (i < 42) {
    println("hello world");
    i = i + 1;
} //will print "hello world"! 42 times.
```

Grammar:

```
WHILE LPAREN expr RPAREN stmt        { While($3, $5) }
```

- loop

The "loop" keyword indicates an infinite loop.

Syntax:

```
let count: int = 0;

loop {
        count += 1;
}
```

Grammar:

```
LOOP stmt { Loop($2) }
```

5h. Block

A block expression conceptually introduces a new namespace scope. Use items can bring new names into scopes and declared items are in scope for only the block itself.

A block will execute each statement sequentially, and then execute the expression (if given). If the block ends in a statement, its value is ().

Syntax:

```
let x: () = { println!("Hello."); }; //a block expression with value ( )
let x: int = { println!("Hello."); 5 }; //a block expression with int value 5.
```

6. Memory and Safety

rusty's main goal is to provide memory safety in the same way as Rust. Declarations without assignment are not allowed.

When variables are assigned to a value, variables are bound to a resource, where they have ownership. As soon as the variable goes out of scope, the variable's resources are automatically freed, and any future references to the variable are no longer valid.

At any given time, there is only one variable that is allowed to own any given resource. After a variable's value has been moved from the old binding to a new one, the old binding can no longer be used. The new binding's value is immutable.

Rather than moving ownership from one variable to another completely, it is possible to borrow ownership via references using the & operator. A new binding that borrows ownership does not deallocate the resource when the binding goes out of scope, allowing one to use the original binding again. Values of references are still immutable.

A mutable reference done via the &mut operator provides a way for mutable ownership borrowing. A * operator is required to access the contents of a mutable reference. A mutable reference can only be made on a variable that is explicitly declared as mutable with the mut qualifier.

Grammar

Scanner

```
{
    open Parser
    let lineno = ref 1
    let depth = ref 0
    let filename = ref ""

    let unescape s =
        Scanf.sscanf ("\"" ^ s ^ "\"") "%S%!" (fun x -> x)
}

let alpha = ['a'-'z' 'A'-'Z']
let escape = '\\' ['\\' ''' '"' 'n' 'r' 't']
let escape_char = ''' (escape) '''
let ascii = ([' '-'!' '#'-'[' ']'-'~'])
let digit = ['0'-'9']
let id = alpha (alpha | digit | '_')*
let string = '"' ( (ascii | escape)* as s) '"'
let char = ''' ( ascii | digit ) '''
let float = (digit+) ['.'] digit+
```

```
let int = digit+
let whitespace = [' ' '\t' '\r']
let newline = '\n'

rule token = parse
  whitespace { token lexbuf }
| newline      { incr lineno; token lexbuf}
| "/*"      { incr depth; comment lexbuf }
| "//"      { comment2 lexbuf }
| '('       { LPAREN }
| ')'       { RPAREN }
| '{'       { LBRACE }
| '}'       { RBRACE }
| ';'       { SEMI }
| ':'       { COLON }
| ','       { COMMA }
(* Operators *)
| '+'       { PLUS }
| '-'       { MINUS }
| '*'       { TIMES }
| '/'       { DIVIDE }
| '%'       { MODULO }
| '='       { ASSIGN }
| "=="      { EQ }
| "!="      { NEQ }
| '<'       { LT }
| "<="      { LEQ }
| ">"       { GT }
| ">="      { GEQ }
| "and"     { AND }
| "or"      { OR }
| "not"     { NOT }
| '.'       { DOT }
| '['       { LBRACK }
| ']'       { RBRACK }
(* Ownership *)
| "&"       { BORROW }
| "mut"     { MUTABLE }
| "&mut"    { MUTABLEBORROW }
(* Branch Control *)
| "if"      { IF }
| "else"    { ELSE }
| "for"     { FOR }
| "while"   { WHILE }
| "loop"    { LOOP }
| "in"      { IN }
(* Data Types *)
| "int"     { INT }
| "float"   { FLOAT }
| "bool"    { BOOL }
| "char"    { CHAR }
| "true"    { TRUE }
| "false"   { FALSE }
| "string"  { STRING }
```

```
| "struct" { STRUCT }
| "impl"   { IMPL }
| "let"    { LET }
| "as"     { AS  }
| "void"   { VOID }
(* function *)
| "fn"     { FUNC }
| "->"     { OUTPUT }
| "return" { RETURN }

(* Other *)
| int as lxm           { INT_LITERAL(int_of_string lxm) }
| float as lxm         { FLOAT_LITERAL(float_of_string lxm) }
| char as lxm          { CHAR_LITERAL( String.get lxm 1 ) }
| escape_char as lxm{ CHAR_LITERAL( String.get (unescape lxm) 1) }
| string               { STRING_LITERAL(unescape s) }
| id as lxm            { ID(lxm) }
| eof                  { EOF }
| '"'                   { raise (Failure("Unmatched quotation at " ^
   string_of_int !lineno)) }
| _ as illegal    { raise (Failure("IllegalCharacter: " ^ Char.escaped
   illegal ^ " at " ^ string_of_int !lineno)) }

and comment = parse
    newline { incr lineno; comment lexbuf }
  |    "*/"       { decr depth; if !depth > 0 then comment lexbuf else token
     lexbuf }
  | "/*"       { incr depth; comment lexbuf }
  |    _          { comment lexbuf }

and comment2 = parse
    newline {token lexbuf}
  | _ {comment lexbuf}
```

Parser

```
/* Ocamlyacc parser for rusty */

%{
open Ast
%}

%token SEMI LPAREN RPAREN LBRACE RBRACE LBRACK RBRACK COMMA COLON DOT
%token PLUS MINUS TIMES DIVIDE MODULO ASSIGN NOT
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR
%token RETURN IF ELSE FOR IN WHILE LOOP AS INT BOOL FLOAT CHAR STRING
%token BORROW MUTABLE MUTABLEBORROW VOID
%token STRUCT IMPL LET
%token FUNC OUTPUT
%token <int> INT_LITERAL
%token <float> FLOAT_LITERAL
%token <char> CHAR_LITERAL
%token <string> STRING_LITERAL
%token <string> ID
```

```
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE MODULO
%right NOT NEG BORROW MUTABLEBORROW DEREF AS
%right RBRACK RPAREN RBRACE
%nonassoc NOACCESS
%left DOT LPAREN LBRACK LBRACE OUTPUT

%start program
%type <Ast.program> program

%%

program:
  decls EOF { $1 }

decls:
    /* nothing */ { [] }
  | decls fdecl   { $2 :: $1 }

fdecl: /* fn foo(x:int) -> int { ... } */
  FUNC ID LPAREN formals_opt RPAREN OUTPUT typ LBRACE stmt_list RBRACE
  {
     {
      fname = $2;
      formals = $4;
    outputType = $7;
      body = List.rev $9
     }
    }

formals_opt:
    /* nothing */ { [] }
  | formal_list   { List.rev $1 }

typ:
      INT   { DataT(IntT) }
  | BOOL  { DataT(BoolT) }
  | FLOAT { DataT(FloatT) }
    | VOID  { DataT(VoidT) }
  | CHAR  { DataT(CharT) }
  | STRING {StringT}
  | LBRACK typ SEMI ID RBRACK          { ArrayTD($2, $4) }
  | LBRACK typ SEMI INT_LITERAL RBRACK { ArrayT($2, $4) } /* [int;10] type is
    an integer array of length 10 */
  | BORROW typ                         { RefT(Immut, $2) } /* &int */
```

```
  | MUTABLEBORROW typ                    { RefT(Mut, $2) } /* &mut int */
  | ID                                   { StructT($1) }

formal_list:
    ID COLON typ                      { [($1, $3)] }
  | formal_list COMMA ID COLON typ { ($3, $5) :: $1 }

struct_list_opt:
  | struct_list { List.rev $1 }

struct_list:
      ID COLON expr                     { [$1, $3] }
  | struct_list COMMA ID COLON expr { ($3, $5) :: $1 }


stmt_list:
  | stmt           { [$1] }
  | stmt_list stmt { $2 :: $1 }

stmt:
    expr SEMI        { Expr $1 }
  | RETURN SEMI      { Return Noexpr }
  | RETURN expr SEMI { Return $2 }
    | LET ID COLON typ ASSIGN expr SEMI         { Declaration(Immut, ($2,
       $4), $6) } /* Assignment */
  | LET MUTABLE ID COLON typ ASSIGN expr SEMI { Declaration(Mut, ($3, $5),
     $7) }
  | LBRACE stmt_list RBRACE                  { Block(List.rev $2) }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7) }
  | FOR LPAREN expr SEMI expr SEMI expr RPAREN stmt { For($3, $5, $7, $9) }
  | LOOP stmt                             { Loop($2) } /* loop {...} is an
     infinite loop */
  | WHILE LPAREN expr RPAREN stmt       { While($3, $5) }
  | STRUCT ID LBRACE formals_opt RBRACE { StructDef($2, $4) } /* Struct */
  | IMPL ID LBRACE fdecl RBRACE         { ImplDef($2, $4) }

expr_list:
      /* nothing */ { [] }
  | expr { [$1] }
  | expr_list COMMA expr { $3 :: $1 }


expr:
    INT_LITERAL      { IntLit($1) } /* Literals */
  | FLOAT_LITERAL    { FloatLit($1) }
  | CHAR_LITERAL     { CharLit($1) }
  | STRING_LITERAL   { StringLit($1) }
  | TRUE             { BoolLit(true) }
  | FALSE            { BoolLit(false) }
  | ID %prec NOACCESS { Id($1) }
  | LBRACK expr_list RBRACK { ArrayLit(List.rev $2) } /* Array */
  | expr LBRACK expr RBRACK { ArrayAccess($1,$3) }
  | LBRACE struct_list_opt RBRACE { StructCreate($2) }
```

```
  | expr DOT ID  { StructAccess($1,$3) } /* Point.x is struct access */
  | expr PLUS   expr { Binop($1, Add,   $3) } /* Binary Operation */
  | expr MINUS  expr { Binop($1, Sub,   $3) }
  | expr TIMES  expr { Binop($1, Mult,  $3) }
  | expr DIVIDE expr { Binop($1, Div,   $3) }
  | expr MODULO expr { Binop($1, Mod,   $3) }
  | expr EQ     expr { Binop($1, Equal, $3) }
  | expr NEQ    expr { Binop($1, Neq,   $3) }
  | expr LT     expr { Binop($1, Less,  $3) }
  | expr LEQ    expr { Binop($1, Leq,   $3) }
  | expr GT     expr { Binop($1, Greater, $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3) }
  | expr AND    expr { Binop($1, And,   $3) }
  | expr OR     expr { Binop($1, Or,    $3) }
  | expr AS     typ  { Cast($1, $3) } /* 32 as float */
  | expr ASSIGN  expr  { Binop($1, Assign, $3)}
  | MINUS expr %prec NEG   { Unop(Neg, $2) } /* Unary Operation */
  | NOT expr               { Unop(Not, $2) }
  | TIMES expr %prec DEREF { Unop(Deref, $2) }
  | BORROW expr            { Unop(Borrow(Immut), $2)}
  | MUTABLEBORROW expr     { Unop(Borrow(Mut), $2)}
  | ID LPAREN actuals_opt RPAREN                  { Call($1, $3) } /* Function
    Call */
  | ID COLON COLON ID LPAREN actuals_opt RPAREN { StructMethodCall($1, $4,
    $6) }   /* Call Struct Methods */
  | LPAREN expr RPAREN { $2 } /* Parenthesis */


actuals_opt:
    /* nothing */ { [] }
  | actuals_list  { List.rev $1 }

actuals_list:
    expr                    { [$1] }
  | actuals_list COMMA expr { $3 :: $1 }
```

Project Plan

In terms of planning our project, we followed the general outline of the course: parser and scanner first, then building the pipeline of translating our rusty program into LLVM code (eg. "Hello World"), then implementing the features of our language. When implementing the features, we started with basic functionality: printing values, binary operations, and control flow. Then came the more complex functionality: structs definitions, struct methods, and arrays. Finally, we tackled memory safety. We followed the advice of our TA to plan our development this way. Once we knew that we had a solid, basic foundation for our language, we could move on to features that are specific to our language (eg. structs, memory safety).

We communicated frequently using our group chat, and met together after class or over the weekend to discuss design of our language as well as to implement our language. Although we delegated specific tasks to each of our team members, we were also actively working together to resolve tough problems. We also met our TA almost weekly to check that we were on track with our project.

One thing that we made sure to do before any coding was clarify the functionality and usage of our language. Everyone had to be on the same page in order for our language to be implemented effectively. We put a lot of effort into understanding Rust and determining which features of Rust we wanted to incorporate into

our language, and which features we would like to remain similar to C. Especially with the mix of Rust and C features in our language, clarification before coding helped us to avoid a lot of misunderstandings during implementation. For example, if two people split the task of implementing a single functionality, with one person writing the semantic checker and another writing the code generation, the two parts of the architecture would be guaranteed to integrate seamlessly since both people had the same understanding of our rusty language.

We also thoroughly looked into projects from previous semesters to understand which features in a language could be implemented in the duration of a semester. We were realistic with our goals from the outset, and this helped us to, for the most part, stick with our proposal from the beginning of the semester. MicroC was an important reference for us to help us get started, and Dice was our inspiration for what an amazing PLT project should look like.

For our design specifications, we knew from the start of our project that we wanted a Rust-like language. Therefore, we mimicked its functionality and syntax. However, for some aspects of our language, such as control flow, we felt that the Rust syntax was quite different from the universally-recognized C syntax, so we incorporated C-like functionality into our language as well.

A practice that we found helpful throughout our development was testing as we developed. Rather than implementing multiple functionalities and testing them all at once, we wrote tests for each feature after we implemented them. This way, we had a full regression suite that checked our overall language functionality after each change is made.

Development Tools

The tools used by everyone in the group were Bitbucket and Sublime Editor. Yanlin and Shiyu used Mac OS Sierra and LLVM 3.9. Emily and Zhuo used Ubuntu 16.04 virtual machines running in VirtualBox and LLVM 3.8. We used Ocaml as our sole development language, and the LLVM API. Other than development tools, we used Facebook Messenger for communication, and Gitbook for our documentation (eg. LRM, final report.)

Programming Style

One challenging aspect in development was that everyone had different programming backgrounds and hence, different programming conventions. It was difficult to determine a style-guide for all of us as it's hard to change our programming habits, but we tried to maintain the following:

- Two spaces for indents
- Lines with 80 characters or less
- Comment code that is not obvious to reader

Timeline

The following was our plan for the semester:

Milestone	Date
Proposal	September 28
Language Reference Manual, Scanner, Parser	October 26
"Hello World"	November 21
Basic functionality (eg. binary operations, control flow)	December 9
Advanced functionality (structs, arrays, memory safety)	December 18
Final report and wrap-up	December 20

Roles

Roles for the team members were specified during the first meeting. As Language Guru, Emily clarified the syntax and functionality for our language. She was the deciding person when we had a conflict on how to implement a feature. As System Architect, Yanlin played a major role in designing the pipeline (from rusty program to LLVM bytecode). As Tester, Zhuo helped add our regression test suite and let us know when certain functionalities did not work. As Manager, Shiyu scheduled milestones and meetings with the TA, and made sure there was constant communication between team members to avoid misunderstandings and conflicts. Although we had separate roles in the team, we often worked together to resolve problems and ended up all contributing to each component of our compiler architecture as well as the test suite.

Project Log

We have been contributing to our project throughout the semester.

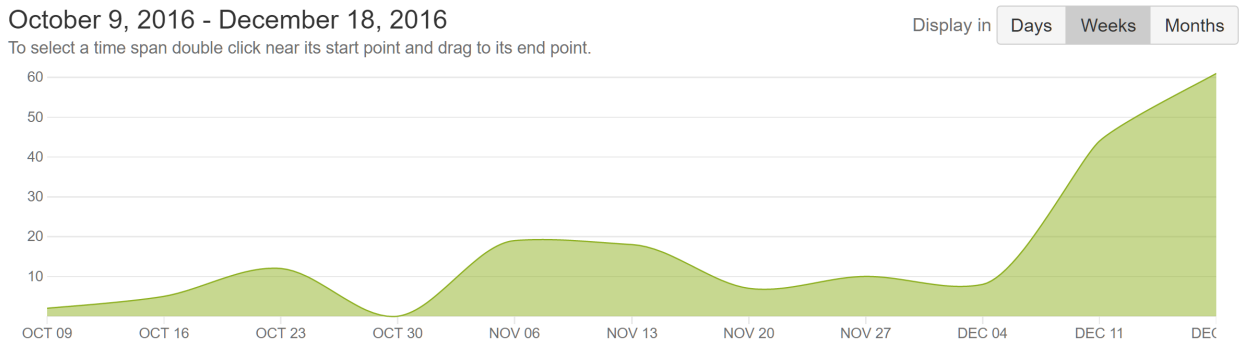

Figure 1: Commit History

commit 6af389b6c1feeb268d25e036f1493cc31b1a24f7 Author: Sherry Qiu sherryq6@gmail.com Date: Tue Dec 20 21:07:20 2016 -0500

```
final
```

commit c37453bd24849665150c77a706448ed8d4536039 Author: Sherry Qiu sherryq6@gmail.com Date: Tue Dec 20 20:08:19 2016 -0500

```
final versions
```

commit e53ff1b2c2dca5de0c37a122b65a67de959faf7b Author: Sherry Qiu sherryq6@gmail.com Date: Tue Dec 20 19:42:26 2016 -0500

```
final codegen
```

commit e33d4d44eb3936e483f0ac5f2b3674c3f34b8ad6 Author: Emily Meng ewm2136@columbia.edu Date: Tue Dec 20 19:23:06 2016 -0500

```
update codegen
```

commit f8a0bc11c457928e6101694cd227c3d01da241e7 Author: Emily Meng ewm2136@columbia.edu Date: Tue Dec 20 19:11:15 2016 -0500

```
cleaning codegen
```

commit e8ee0c91957046f28dd73ec7e31e9c6eded8812f Author: Emily Meng ewm2136@columbia.edu Date: Tue Dec 20 19:00:34 2016 -0500

```
clean semant.ml
```

commit ede5061b0cf5c3647d0f93772870ed106bdc8a7a Author: Emily Meng ewm2136@columbia.edu Date: Tue Dec 20 18:49:39 2016 -0500

```
cleaned pre_semant.ml
```

commit e55854ff08e4eded4f6b31d2502a9d4bad7692ba Author: Emily Meng ewm2136@columbia.edu Date: Tue Dec 20 18:47:32 2016 -0500

```
clean ast.ml and sast.ml
```

commit bf758e954be88b683f6fd2743a1e63d0071172d3 Author: Emily Meng ewm2136@columbia.edu Date: Tue Dec 20 18:43:23 2016 -0500

```
removed pos
```

commit 99e14c880f223cd65d493c7c53d31049322bc17b Author: Emily Meng ewm2136@columbia.edu Date: Tue Dec 20 18:35:03 2016 -0500

```
cleaning parser and scanner
```

commit a50a01f084c3b60444f10a9b8edb52c71abfac22 Author: Sherry Qiu sherryq6@gmail.com Date: Tue Dec 20 18:10:52 2016 -0500

```
rm tuples
```

commit 6b6dbea77ff9ab2fc128c204ede40469ef5c5251 Author: alan97 duanyanl97@gmail.com Date: Tue Dec 20 03:32:45 2016 -0500

```
added AST SAST graphs
```

commit a59cdb5eeb0a9b75c630be496367d041349fe6f7 Author: alan97 duanyanl97@gmail.com Date: Mon Dec 19 17:50:33 2016 -0500

```
makes slides look better XD
```

commit 7c984d65850e187d1fe05148ec0753293123ad84 Author: zk2202 zk2202@columbia.edu Date: Mon Dec 19 17:13:42 2016 -0500

```
fix tests
```

commit 46ef96cdac9b7b7705985065d5e58ea71c1fc777 Author: zk2202 zk2202@columbia.edu Date: Mon Dec 19 17:12:49 2016 -0500

```
testcase fixed
```

commit eb38aabc6e3a76a130f77dc286592e708ea1d9a9 Author: Sherry Qiu sherryq6@gmail.com Date: Mon Dec 19 17:07:04 2016 -0500

```
update struct demo
```

commit 193566394b6eaff4583aa31314d59a04983cf623 Author: alan97 duanyanl97@gmail.com Date: Mon Dec 19 16:54:37 2016 -0500

```
everything looks pretty nice xd
```

commit cbf2d8cac03be90bd9406bd09838a491741863cd Author: alan97 duanyanl97@gmail.com Date: Mon Dec 19 16:45:03 2016 -0500

```
add string
```

commit 0e74c873f342a8f19bdb884cf6736c21f5cfdf15 Merge: 9c61a7c f8338f4 Author: alan97 duanyanl97@gmail.com Date: Mon Dec 19 16:11:05 2016 -0500

```
add impl?
```

commit 9c61a7c64fff2a1893ad48309587bc96ce7c253e Author: alan97 duanyanl97@gmail.com Date: Mon Dec 19 16:09:49 2016 -0500

```
adding impl?
```

commit f8338f44dddbb7179fc05d490b25ea905aa9183e Merge: bc7b38f 98b0bdd Author: Emily Meng ewm2136@columbia.edu Date: Mon Dec 19 16:08:45 2016 -0500

```
Merge branch 'addSAST' of https://bitbucket.org/sherryq/rusty into addSAST
```

commit bc7b38f773030eb3c348c01d74eaaa00233ece17 Author: Emily Meng ewm2136@columbia.edu Date: Mon Dec 19 16:08:05 2016 -0500

```
removed warning
```

commit 98b0bdd07f09b418d6ddee17edd91a93d7c23cf6 Author: Sherry Qiu sherryq6@gmail.com Date: Mon Dec 19 16:07:33 2016 -0500

```
update struct demo, add mem demos
```

commit 61ae2c944ae108d9c6b9ba70657b62c3ac4364c0 Author: Emily Meng ewm2136@columbia.edu Date: Mon Dec 19 16:00:12 2016 -0500

```
removed almost all warnings from semant.ml
```

commit 2fd6dbe3db9dd8f4a810d6baf4a50d3ed72ac11e Author: Emily Meng ewm2136@columbia.edu Date: Mon Dec 19 15:44:18 2016 -0500

```
removed codegen.ml warnings
```

commit f0025a8c637697072431ac341f54c2815a4fb9e3 Author: Emily Meng ewm2136@columbia.edu Date: Mon Dec 19 15:44:06 2016 -0500

```
removed ast.ml warnings
```

commit cc93cf450208ba16d89d5b48f36cb0b0fd73099b Author: Sherry Qiu sherryq6@gmail.com Date: Mon Dec 19 15:05:19 2016 -0500

```
some warnings cleared in semant
```

commit b1ac4ecd83a1c209ae2b50e2e5bbb49e30f3bd23 Merge: 07743da e932158 Author: alan97 duanyanl97@gmail.com Date: Mon Dec 19 15:02:28 2016 -0500

```
Merge branch 'addSAST' of https://bitbucket.org/sherryq/rusty into addSAST
```

commit 07743daff78edc15420ab9f7f90055e7498cbe80 Author: alan97 duanyanl97@gmail.com Date: Mon Dec 19 15:02:22 2016 -0500

```
fixed demo2
```

commit e9321583bfe6ab832b5c42e42ff0933b9c5e4ccf Author: Sherry Qiu sherryq6@gmail.com Date: Mon Dec 19 14:49:57 2016 -0500

```
remove warnings in presemant
```

commit f36e8dcac90aabdc841e28982ab1491f321eeb32 Merge: 9614dd3 8c7f8b6 Author: alan97 duanyanl97@gmail.com Date: Mon Dec 19 14:47:01 2016 -0500

```
fixed demo1
```

commit 9614dd37083682d6ea1a287a99f5fe2a045ca52a Author: alan97 duanyanl97@gmail.com Date: Mon Dec 19 14:39:05 2016 -0500

```
fixed return
```

commit 8c7f8b6389bae4df70c05fb43c0a074be345120b Author: Sherry Qiu sherryq6@gmail.com Date: Mon Dec 19 13:53:00 2016 -0500

```
demo files
```

commit 290ef5ffe4a2165e73ef276311b52d1b4e34795a Author: Sherry Qiu sherryq6@gmail.com Date: Mon Dec 19 11:17:31 2016 -0500

```
final presentation
```

commit ddd26be7bc6cefb2654544c3d78fcb67ffbdae55 Author: zk2202 zk2202@columbia.edu Date: Mon Dec 19 10:25:40 2016 -0500

```
debug
```

commit 181a3a65514271fbdcbabe8f82cfcbb6d20eefb0 Author: Sherry Qiu sherryq6@gmail.com Date: Mon Dec 19 01:56:16 2016 -0500

```
clear warnings in codegen
```

commit f560213acc26dc2f0cf0ca34fe8e3eae39835952 Author: alan97 duanyanl97@gmail.com Date: Mon Dec 19 01:10:48 2016 -0500

```
test case wrong, code works
```

commit d98c36c58d84e9f6861d1d34afbbb9414b978fe8 Merge: 0cab1b2 a3bc17e Author: alan97 duanyanl97@gmail.com Date: Mon Dec 19 01:02:28 2016 -0500

```
Merge branch 'addSAST' of https://bitbucket.org/sherryq/rusty into addSAST
```

commit 0cab1b2084aab98fcd32bc27d4f83fa781e1193d Author: alan97 duanyanl97@gmail.com Date: Sun Dec 18 23:35:37 2016 -0500

```
fixed struct to function
```

commit a3bc17e86bf7261c25ea2932847dd870412f7530 Merge: 4097553 b07f05f Author: zk2202 zk2202@columbia.edu Date: Sun Dec 18 23:31:37 2016 -0500

```
debug
Merge branch 'addSAST' of https://bitbucket.org/sherryq/rusty into addSAST
```

commit 40975539905334708c3df3399b90e600faeac180 Author: zk2202 zk2202@columbia.edu Date: Sun Dec 18 23:25:04 2016 -0500

```
debug
```

commit 6d39ec5dc72db2dc4d853141cb1acf1737353070 Author: Emily Meng ewm2136@columbia.edu Date: Sun Dec 18 23:17:27 2016 -0500

```
need to fix pre_semant.ml
```

commit b07f05f7ba43a54c12f8775147d461fd854f68f6 Author: alan97 duanyanl97@gmail.com Date: Sun Dec 18 22:04:05 2016 -0500

```
changed syntax for impl call
```

commit 2a6f9fc59ee5500c9c91078cc2d0ffaa46dc31cf Author: alan97 duanyanl97@gmail.com Date: Sun Dec 18 21:23:21 2016 -0500

```
so far everything builds
```

commit ad5fc69e1d2fe9d74c077d4a6306aeea3b885188 Author: alan97 duanyanl97@gmail.com Date: Sun Dec 18 21:16:02 2016 -0500

```
solving
```

commit ca2805f39278428b8312a6b2541863423aa00361 Author: alan97 duanyanl97@gmail.com Date: Sun Dec 18 21:15:48 2016 -0500

```
solving
```

commit 77401291c283bf6441ba40410ae78513534cb42d Author: alan97 duanyanl97@gmail.com Date: Sun Dec 18 21:06:10 2016 -0500

```
solving
```

commit 3c67127bd290e3df5683172f62da42cda7e05442 Author: alan97 duanyanl97@gmail.com Date: Sun Dec 18 20:49:32 2016 -0500

```
struct parser solved
```

commit 5f774f4edbae16acff93c8f3c499fb4ccd582cab Author: alan97 duanyanl97@gmail.com Date: Sun Dec 18 20:45:33 2016 -0500

```
removed test_fail_blablabla
```

commit e2301e43cf36a2940f43ed14dcf117b3d71696a2 Author: alan97 duanyanl97@gmail.com Date: Sun Dec 18 20:45:09 2016 -0500

```
some tests failed?
```

commit 9089cdc243091d00c36dde1c61ea1859dc50b78f Merge: 7d8a5b9 0d39fd5 Author: alan97 duanyanl97@gmail.com Date: Sun Dec 18 20:43:16 2016 -0500

```
resolved
```

commit 7d8a5b97ee48860d837487be512b40364f43e939 Author: alan97 duanyanl97@gmail.com Date: Sun Dec 18 20:40:30 2016 -0500

```
working
```

commit 0d39fd5a6b03f1afc14c577204c1c1af068fa9d6 Author: Emily Meng ewm2136@columbia.edu Date: Sun Dec 18 20:39:38 2016 -0500

```
updated parser
```

commit cc4273b769ef18f9fb44837db00162aa7375a71f Author: Emily Meng ewm2136@columbia.edu Date: Sun Dec 18 19:35:05 2016 -0500

```
updated pre_semant.ml
```

commit 6717b86e38af4b8c4f8c0782b7b69abf0fa1ae11 Author: Sherry Qiu sherryq6@gmail.com Date: Sun Dec 18 19:04:44 2016 -0500

```
support strings
```

commit d2bf3c6fdc146517116232f287f3ca4b38ac8f7b Author: alan97 duanyanl97@gmail.com Date: Sun Dec 18 18:30:27 2016 -0500

```
fixed syntax error of presemant
```

commit 54a3ab4170d8e7840fb193f1429550798978650b Author: Emily Meng ewm2136@columbia.edu Date: Sun Dec 18 18:08:08 2016 -0500

```
added preprocessing to change impldef and structmethodcall in pre_semant.ml
```

commit 9478f32bc5cdce4361b5a502a4f5c5d2767e3404 Author: alan97 duanyanl97@gmail.com Date: Sun Dec 18 17:49:49 2016 -0500

```
checked memory safety
```

commit c435eafe58a7ca58148490abb5d46494f9175278 Author: Sherry Qiu sherryq6@gmail.com Date: Sun Dec 18 17:25:35 2016 -0500

```
add new struct tests
```

commit 3df4848f692a2fca94a8f14db5e1ab4387451d27 Author: Sherry Qiu sherryq6@gmail.com Date: Sun Dec 18 17:03:36 2016 -0500

```
update struct tests
```

commit 0a1a678983a8732a15fd10e3dc7ac901282b4e66 Author: Sherry Qiu sherryq6@gmail.com Date: Sun Dec 18 16:34:53 2016 -0500

```
add struct tests
```

commit d4c2301330a1c31ac3e48ff774a03d06fd449d67 Author: alan97 duanyanl97@gmail.com Date: Sun Dec 18 15:44:58 2016 -0500

```
more memory safety check
```

commit 241c8b7d8887da0c2e9cd00b2ab747dc94813426 Author: alan97 duanyanl97@gmail.com Date: Sun Dec 18 14:53:50 2016 -0500

```
struct create and access now work as expected
```

commit 96982926acaa4eb831f67d54028d711f965df5f0 Author: zk2202 zk2202@columbia.edu Date: Sun Dec 18 11:18:08 2016 -0500

```
debug
```

commit 493ed5c4e553549b9d00178e5ab3cc17ac3839b3 Author: zk2202 zk2202@columbia.edu Date: Sun Dec 18 11:17:34 2016 -0500

```
debug
```

commit aa597004785d1530d96ca5d92df3acd4a882e2ca Merge: 991871b 1cedd84 Author: zk2202 zk2202@columbia.edu Date: Sun Dec 18 10:55:07 2016 -0500

```
retry
Merge branch 'addSAST' of https://bitbucket.org/sherryq/rusty into addSAST
```

commit 991871b14b57356e75a27ba8d4aa5bd64f3d7842 Author: zk2202 zk2202@columbia.edu Date: Sun Dec 18 09:48:28 2016 -0500

debug

commit 1cedd84dfcc5d6dd5b3c815b6a9933bf0d1504a1 Author: Sherry Qiu sherryq6@gmail.com Date: Sun Dec 18 03:49:48 2016 -0500

origin var not in varmap

commit 12a27646138d52c96153ee1fe79e1043ae61e25a Author: Sherry Qiu sherryq6@gmail.com Date: Sun Dec 18 02:56:56 2016 -0500

struct access builds

commit 6513b9598287a89609eca58c474070455519860d Author: Sherry Qiu sherryq6@gmail.com Date: Sun Dec 18 01:55:59 2016 -0500

struct create supports all types

commit 56eafe2e172aec6960b473ed25604a66abc62498 Author: Sherry Qiu sherryq6@gmail.com Date: Sun Dec 18 01:43:29 2016 -0500

struct instance definition works

commit cbea783470c1daf86017d49d249dcec10318cbfa Author: Sherry Qiu sherryq6@gmail.com Date: Sun Dec 18 01:26:28 2016 -0500

why is this Ptr type error back again

commit 9a1e2347c393981528fbd19378575586d463fe9f Merge: 19f45e9 ef1d536 Author: Emily Meng ewm2136@columbia.edu Date: Sun Dec 18 01:21:20 2016 -0500

merge resolve

commit 19f45e9cff8c1c452e23700dc036a3bfeb1e09c9 Author: Emily Meng ewm2136@columbia.edu Date: Sun Dec 18 01:15:51 2016 -0500

fix?

commit ef1d536fc37cc528c69f41be1bc377fcbfb80460 Merge: 1f8d12a 5b4546f Author: alan97 duanyanl97@gmail.com Date: Sun Dec 18 01:02:30 2016 -0500

memory basically works

commit 1f8d12aae9ba4a6c2e4eef455e43a0c129bc31f9 Author: alan97 duanyanl97@gmail.com Date: Sun Dec 18 01:01:31 2016 -0500

memory basically works

commit 5b4546fb265675d1c9215a3f5242e99530d0f716 Author: Emily Meng ewm2136@columbia.edu Date: Sun Dec 18 00:55:22 2016 -0500

fixes to structaccess in semant.ml and test for structaccess

commit b86df1e66bdf3efc2b2ce9946f29ee672f574e0c Author: Sherry Qiu sherryq6@gmail.com Date: Sat Dec 17 22:15:44 2016 -0500

rm out files in Makefile

commit 84e889992585569fdf83f5d51c4ca0e8f4753c2c Merge: d13d934 40c1622 Author: Sherry Qiu sherryq6@gmail.com Date: Sat Dec 17 22:09:46 2016 -0500

```
Merge branch 'addSAST' of https://bitbucket.org/sherryq/rusty into addSAST
```

commit 40c1622aa7b972fd9f0c7611a0fb1d6ddd272d3c Author: Emily Meng ewm2136@columbia.edu Date: Sat Dec 17 22:06:56 2016 -0500

```
remove .out files
```

commit d13d93484d2262f024b13403ad3c21b1fea8f204 Author: Sherry Qiu sherryq6@gmail.com Date: Sat Dec 17 22:04:39 2016 -0500

```
ptr does not point to type Value
```

commit 4ac0e30ea3445ee31719a8ea2f8d146a40ace8dc Author: alan97 duanyanl97@gmail.com Date: Sat Dec 17 20:48:27 2016 -0500

```
borrow, deref works
```

commit 5a2aab84487eb22b4b13c506bac2fcbb9ccb58cd Merge: 316b53a effbd54 Author: alan97 duanyanl97@gmail.com Date: Sat Dec 17 19:14:36 2016 -0500

```
Merge branch 'addSAST' of https://bitbucket.org/sherryq/rusty into addSAST
```

commit 316b53aab5cebf7132e76bcf13b3dcce70c84d04 Author: alan97 duanyanl97@gmail.com Date: Sat Dec 17 19:14:32 2016 -0500

```
added borrow
```

commit effbd5452ae34f4ecd73c9a6f286e4ca30b9aaea Merge: f06b7fd bc7b5c6 Author: zk2202 zk2202@columbia.edu Date: Sat Dec 17 19:09:39 2016 -0500

```
Merge branch 'addSAST' of https://bitbucket.org/sherryq/rusty into addSAST
```

commit f06b7fd6ae28298b74ebc2a1946e944015eb03eb Merge: dec309b b6c88c2 Author: zk2202 zk2202@columbia.edu Date: Sat Dec 17 19:09:18 2016 -0500

```
fix tests
```

commit bc7b5c6502a5b6c829541f2b1982647b89c6e30c Author: Emily Meng ewm2136@columbia.edu Date: Sat Dec 17 19:04:25 2016 -0500

```
catch type mismatches in semant.ml
```

commit dec309bb8ad0feaaa509f8c55973cad6438bdfde Author: zk2202 zk2202@columbia.edu Date: Sat Dec 17 19:00:09 2016 -0500

```
debug
```

commit b6c88c23464ff313ff3283fc3c59435b98274879 Author: Emily Meng ewm2136@columbia.edu Date: Sat Dec 17 18:41:54 2016 -0500

```
updated struct semant.ml
```

commit 2d794045955be93dcfe5210e572a093c40994880 Author: Emily Meng ewm2136@columbia.edu Date: Sat Dec 17 16:40:55 2016 -0500

```
tests for struct semant
```

commit 77350373e147e604fc0a0cbdfc6b3b325c8da5ef Author: Emily Meng ewm2136@columbia.edu Date: Sat Dec 17 16:38:10 2016 -0500

```
updated struct stuf in semant.ml
```

commit 9d6274156fa3d478dd9303ebdec91414da5f9109 Author: alan97 duanyanl97@gmail.com Date: Sat Dec 17 16:08:04 2016 -0500

```
merged with borrow checker
```

commit acb942eca7cc308319fa1451f762b434e3d3cbc5 Merge: 56a85dd af6851d Author: alan97 duanyanl97@gmail.com Date: Sat Dec 17 15:54:22 2016 -0500

```
merged with borrow checker
```

commit 56a85dd29265974d3f6401fe4432a6894f582491 Author: alan97 duanyanl97@gmail.com Date: Sat Dec 17 15:19:40 2016 -0500

```
trying to fix test cases @.@
```

commit cb141e2042eb4db918e0564586b3335bcbc54ef6 Author: alan97 duanyanl97@gmail.com Date: Sat Dec 17 15:15:52 2016 -0500

```
testall fixed
```

commit 1822b007ef6ecdb9a88342ddf5671083b722d5ab Author: alan97 duanyanl97@gmail.com Date: Sat Dec 17 15:07:07 2016 -0500

```
changed changepy
```

commit 10698d7acbd62b3a145c9353db3ba27e640e140f Author: zk2202 zk2202@columbia.edu Date: Sat Dec 17 14:57:25 2016 -0500

```
uploaded again
```

commit 738cc4605bb9a5d05ec1d9272e4cd130c395d658 Merge: d1716cb 0d88e3c Author: zk2202 zk2202@columbia.edu Date: Sat Dec 17 14:50:18 2016 -0500

```
use change.py to fix tests
```

commit d1716cb2da9e23d09ba21ab3adc3bb1a281d2010 Merge: cdafea2 a05d4c4 Author: zk2202 zk2202@columbia.edu Date: Sat Dec 17 14:47:02 2016 -0500

```
fix main int
```

commit 0d88e3c055ff86279385451a91c2b114ec6b891e Author: Emily Meng ewm2136@columbia.edu Date: Sat Dec 17 14:40:02 2016 -0500

```
updated test script to go into correct directories
```

commit cdafea23b856c1740af665fe56aa7532c0c51aa0 Author: zk2202 zk2202@columbia.edu Date: Sat Dec 17 14:39:44 2016 -0500

```
fix main int
```

commit a05d4c42c5d382cae5157a827570f8c7e9c8209a Author: alan97 duanyanl97@gmail.com Date: Sat Dec 17 14:37:35 2016 -0500

```
array fixed, parser fixed
```

commit 09ed2d2b0e95eadacbf35cf5d26df8c74f413448 Author: alan97 duanyanl97@gmail.com Date: Sat Dec 17 14:37:10 2016 -0500

```
array fixed, parser fixed
```

commit c0282daa9ff02b87b2128915df972c857f519a11 Merge: 70dd153 387baba Author: Emily Meng ewm2136@columbia.edu Date: Fri Dec 16 22:42:29 2016 -0500

```
Merge branch 'addSAST' of https://bitbucket.org/sherryq/rusty into addSAST
```

commit 70dd153617b565beeb7c8aff70d865a0c08b2c23 Author: Emily Meng ewm2136@columbia.edu Date: Fri Dec 16 22:41:52 2016 -0500

```
structcreate and struct for declaration compiles in semant.ml
```

commit 387babab6351e01378b0b037341f399503eade6c Author: Sherry Qiu sherryq6@gmail.com Date: Fri Dec 16 21:58:33 2016 -0500

```
fix env for arrays, create non-integer arrays
```

commit 594c5d9afd50c0650d091f4277935193b9d55fa8 Author: Emily Meng ewm2136@columbia.edu Date: Fri Dec 16 21:43:34 2016 -0500

```
finally fix merge conflict
```

commit 305fe1fa59d24e2adf78fe1e2b561d175854cd87 Merge: 3719bf6 173da78 Author: Emily Meng ewm2136@columbia.edu Date: Fri Dec 16 21:42:10 2016 -0500

```
finish merge
```

commit 3719bf65724834081794e95cc202b4519a8c042d Author: Emily Meng ewm2136@columbia.edu Date: Fri Dec 16 21:40:22 2016 -0500

```
trying to get rid of merge conflic
```

commit 173da786dabac7d167dcde43c35a2a3dc7ebad3b Author: Sherry Qiu sherryq6@gmail.com Date: Fri Dec 16 19:55:10 2016 -0500

```
function and control flow tests
```

commit 2aa41ea9d767357180ff00c5577c304f1a89546c Author: Sherry Qiu sherryq6@gmail.com Date: Fri Dec 16 19:34:50 2016 -0500

```
if, for, while work
```

commit 5feff337e853f64915614891ed4ed5fa1b6d904c Author: Emily Meng ewm2136@columbia.edu Date: Fri Dec 16 19:22:09 2016 -0500

```
structdef works in semant.ml
```

commit c2aab05f84e91a7f37eb51c81895973c2119e623 Author: Emily Meng ewm2136@columbia.edu Date: Fri Dec 16 19:06:49 2016 -0500

```
removed const and static
```

commit f554fd644426a20508415eeb559f977f2829745e Author: Emily Meng ewm2136@columbia.edu Date: Fri Dec 16 18:48:53 2016 -0500

```
updated struct create and semant.ml
```

commit 170768b289e5a09a7fe28f65586072097743b360 Author: alan97 duanyanl97@gmail.com Date: Fri Dec 16 18:06:24 2016 -0500

```
fixed binop geq, eq, le,leq
```

commit 1c1081b9573e79647b5263431ae8bcd772c2c0ee Merge: 6a9179f 71e41dc Author: alan97 duanyanl97@gmail.com Date: Fri Dec 16 17:53:52 2016 -0500

```
Merge branch 'addSAST' of https://bitbucket.org/sherryq/rusty into addSAST
```

commit 6a9179fb00d31d6cf326ae089a86fa1e92c008ae Merge: 333f769 4ce7c54 Author: alan97 duanyanl97@gmail.com Date: Fri Dec 16 17:53:44 2016 -0500

```
new
```

commit 71e41dc120664196ecd615b1ed91e0f9d5a0cec6 Author: Zhuo Kong zk2202@columbia.edu Date: Fri Dec 16 22:50:51 2016 +0000

```
codegen.ml edited online with Bitbucket
```

commit 333f769d7d643d7f1675d146480dbab19b053df6 Author: alan97 duanyanl97@gmail.com Date: Fri Dec 16 17:49:09 2016 -0500

```
array fixed
```

commit 4ce7c54cbae4792d8f347fcfa39b8f6e0651d09d Author: zk2202 zk2202@columbia.edu Date: Fri Dec 16 17:36:59 2016 -0500

```
binop
```

commit 1ab266e5f0ffc0a107b93ed6587820596b17abc7 Merge: b1b4d56 0a27c94 Author: Emily Meng ewm2136@columbia.edu Date: Fri Dec 16 15:00:48 2016 -0500

```
Merge branch 'addSAST' of https://bitbucket.org/sherryq/rusty into addSAST
```

commit b1b4d56a02cc0cffa6729f797a27fecd26e4c5aa Author: Emily Meng ewm2136@columbia.edu Date: Fri Dec 16 14:59:53 2016 -0500

```
removed unused variables warnings in codegen.ml
```

commit 0a27c94188e3b449f2fc7aa5339a2d390d38920c Author: Sherry Qiu sherryq6@gmail.com Date: Fri Dec 16 14:45:06 2016 -0500

```
add break
```

commit 510391be4393e715d7a1d97649fd2fa22c843426 Author: Sherry Qiu sherryq6@gmail.com Date: Fri Dec 16 14:10:25 2016 -0500

```
array pointer type cast issues
```

commit af6851d2478de2b384b166059c7e7da784ff8ced Author: alan97 duanyanl97@gmail.com Date: Fri Dec 16 14:00:36 2016 -0500

```
first stage -- now detect use of moved value
```

commit 3b0ea8f6642f3ee68ff1f195fd5905da93ebd6a5 Author: alan97 duanyanl97@gmail.com Date: Fri Dec 16 03:59:34 2016 -0500

```
code gen scoping now work
```

commit 2a02527ef1b9bf4de427a4c1fccf6f3c9b4a9189 Author: alan97 duanyanl97@gmail.com Date: Fri Dec 16 03:10:32 2016 -0500

```
code gen scoping
```

commit 75a66bcf017afb93e00a1061a13d2d1c8f18096f Author: alan97 duanyanl97@gmail.com Date: Fri Dec 16 02:34:16 2016 -0500

```
scoping semant check passed
```

commit ad46d8340bd277bc6705ae7264b6c684407a6045 Author: alan97 duanyanl97@gmail.com Date: Fri Dec 16 02:18:41 2016 -0500

```
added scoping
```

commit 8e337ea76433fd8ee1d11a01bf1d86d25d195479 Author: alan97 duanyanl97@gmail.com Date: Fri Dec 16 01:30:04 2016 -0500

```
updated
```

commit e108469432212e014035261c2d131cddafb0c791 Author: alan97 duanyanl97@gmail.com Date: Fri Dec 16 01:26:25 2016 -0500

```
merged with sast
```

commit c33a7f2ad6fc511e49455c3b0fdcf264f6cc99ab Author: Emily Meng ewm2136@columbia.edu Date: Fri Dec 16 00:17:43 2016 -0500

```
structdef and impldef compile but need more checks in semant.ml
```

commit 86ba6c3eb2e50c99c8249d48364f2ed081169008 Author: Emily Meng ewm2136@columbia.edu Date: Thu Dec 15 22:49:25 2016 -0500

```
added structdef and impldef along with helper functions to semant.ml
```

commit 9d4b2da8b8caa12bfb8f6a6749f9c224560b85ac Author: alan97 duanyanl97@gmail.com Date: Tue Dec 13 21:03:24 2016 -0500

```
function return, function actuals fixed
```

commit 69059f7201eddaade879af8bd06a9b6881325c15 Author: alan97 duanyanl97@gmail.com Date: Tue Dec 13 19:38:10 2016 -0500

```
fixed function formal not found err
```

commit 99023f9ff6ed3aaf656bbf34c33a0b278871d9c2 Author: Sherry Qiu sherryq6@gmail.com Date: Tue Dec 13 18:36:14 2016 -0500

```
changed test_func1
```

commit 379f1b8d26d98f8ca45f37f0dbe1c772fd0b840b Author: alan97 duanyanl97@gmail.com Date: Tue Dec 13 18:24:15 2016 -0500

```
array creation done
```

commit 6aa6675e308931cec64e4386665d586b34356e2e Author: Emily Meng ewm2136@columbia.edu Date: Tue Dec 13 18:23:20 2016 -0500

```
removed old directories for tests
```

commit f08a3963c7d3958230bd2fd8877ff157f8b63dd6 Author: Emily Meng ewm2136@columbia.edu Date: Tue Dec 13 18:19:57 2016 -0500

```
rename directories for tests
```

commit 1ebd6ae51147753d48ddcd7dbf73b59d949f6ca2 Author: Emily Meng ewm2136@columbia.edu Date: Tue Dec 13 18:12:31 2016 -0500

```
update test script directory
```

commit 9334b5bb4f75053bcb603d1b69240a5c0461a86a Author: Sherry Qiu sherryq6@gmail.com Date: Fri Dec 9 21:17:29 2016 -0500

```
add array access
```

commit 657aaca6991021d61b41593a140a58be04eb0a7d Author: Sherry Qiu sherryq6@gmail.com Date: Fri Dec 9 20:53:22 2016 -0500

```
array def builds
```

commit 91669b9c6d1fd38afdcfa967202624a4ab49f825 Author: Sherry Qiu sherryq6@gmail.com Date: Fri Dec 9 20:10:24 2016 -0500

```
array init
```

commit ba256bbe224909e58ec769c819a0d600d0d72c13 Author: alan97 duanyanl97@gmail.com Date: Fri Dec 9 17:23:35 2016 -0500

```
variable first-round tested.
```

commit f9865a9036df026007d9aabbd780a08e869252ef Author: alan97 duanyanl97@gmail.com Date: Thu Dec 8 13:40:12 2016 -0500

```
variable added
```

commit a9da426a02bf43887d72131cf04e63d2c47c1330 Merge: 3ff2bfc 5fd7c4f Author: alan97 duanyanl97@gmail.com Date: Thu Dec 8 13:23:07 2016 -0500

```
variable added:
```

commit 3ff2bfc3957bee3fd77ecec987d7047d1015aef3 Author: alan97 duanyanl97@gmail.com Date: Thu Dec 8 13:21:45 2016 -0500

```
variable added:
```

commit 5fd7c4f7267b276cf260f3c9bd700545fabd99f0 Author: zk2202 zk2202@columbia.edu Date: Thu Dec 8 05:49:42 2016 -0500

```
changing tests. tests about function are in tests/notimplented/function. in
    order to use them just paste them to  /tests and run ./testall.sh.
tests in /notimplented means we know what to do to pass them and tests in
    fail mean that we should fix the code.
```

commit 90698aaed339852b1a969d87f2f28114f6a11030 Author: Sherry Qiu sherryq6@gmail.com Date: Thu Dec 8 02:34:06 2016 -0500

```
add functions
```

commit def4baa1c80f5fdde4c4e14235d05fcfb1c3c795 Author: Emily Meng ewm2136@columbia.edu Date: Wed Nov 30 21:32:13 2016 -0500

```
fixed while and added some unop
```

commit d87e9798c0fb635dac9639719b8ecb5410f2e276 Author: Sherry Qiu sherryq6@gmail.com Date: Wed Nov 30 21:13:29 2016 -0500

```
add while test
```

commit e78f431eccbf0be23d1492b5df4be8ab33d2fd6a Author: alan97 duanyanl97@gmail.com Date: Wed Nov 30 20:15:39 2016 -0500

```
fixed add and if
```

commit 4f9dadd50455b85d1f306db98ec70aba8828b1f1 Author: alan97 duanyanl97@gmail.com Date: Wed Nov 30 20:14:58 2016 -0500

```
fixed if and add
```

commit d5328d3800c74a2b95651ab3375088d8d46cec59 Author: Sherry Qiu sherryq6@gmail.com Date: Wed Nov 30 19:53:44 2016 -0500

```
print int, some arith ops work
```

commit 629920701f184667e952d7ad28938d5471ab6cbf Author: alan97 duanyanl97@gmail.com Date: Wed Nov 30 19:25:45 2016 -0500

```
fixed print, SIf, SWhile
```

commit 7a8fa6257dd22a2853f6fa4087bd1ca44446da68 Author: Sherry Qiu sherryq6@gmail.com Date: Sat Nov 26 14:10:12 2016 -0500

```
semantic checking for if and while
```

commit d59b228d37060a5df43727ae270e90bf76976636 Author: Sherry Qiu sherryq6@gmail.com Date: Fri Nov 25 22:30:51 2016 -0500

```
modify expr builder
```

commit b1e3d9d2e67903a565a645e405367f842ecf4bb3 Author: Sherry Qiu sherryq6@gmail.com Date: Fri Nov 25 20:21:47 2016 -0500

```
get rid of println semant logic
```

commit cf2b9768543c111d52ac3a8db564a4aff06d55ea Author: Sherry Qiu sherryq6@gmail.com Date: Sat Nov 19 23:28:23 2016 -0500

```
actually fix some arith tests
```

commit 96e162b8537b55ae9a58b531ad8f70d8b34549fd Author: Sherry Qiu sherryq6@gmail.com Date: Sat Nov 19 23:27:06 2016 -0500

```
fixed some arith tests
```

commit bc0eb59e08465bcef6c8b0f33546df0fc4bfea4d Merge: d2167ec 7499341 Author: alan97 duanyanl97@gmail.com Date: Thu Nov 17 21:47:03 2016 -0500

```
Merge branch 'addSAST' of https://bitbucket.org/sherryq/rusty into addSAST
```

commit d2167ecff08db3e91edcf53cfe18b98e409634a6 Author: alan97 duanyanl97@gmail.com Date: Thu Nov 17 21:46:05 2016 -0500

```
testing gitignore
```

commit 7499341019a263eee017ae52530825ac6d906900 Author: Sherry Qiu sherryq6@gmail.com Date: Thu Nov 17 21:44:34 2016 -0500

```
update make clean
```

commit 2625109c5118ce1c3475397b0a1ec49da546901b Author: alan97 duanyanl97@gmail.com Date: Thu Nov 17 21:40:59 2016 -0500

```
parser fixed
```

commit bae341c37aae0ae61ebe502f53832336cb75584f Author: alan97 duanyanl97@gmail.com Date: Thu Nov 17 21:23:06 2016 -0500

```
added semant and codegen
```

commit e2cabfa270593b3241ad38e83e11377309eb4625 Author: alan97 duanyanl97@gmail.com Date: Thu Nov 17 21:21:58 2016 -0500

```
should work...?
```

commit 48a944658985853eb7285eef6cafbb700b6782c7 Author: alan97 duanyanl97@gmail.com Date: Thu Nov 17 21:13:01 2016 -0500

```
almost finished
```

commit 07b17b9543564890e4957259d4af503a180facca Author: alan97 duanyanl97@gmail.com Date: Thu Nov 17 20:23:55 2016 -0500

```
working
```

commit ad7eda5e603d85d426ad524a5089bb30c5cb0796 Author: alan97 duanyanl97@gmail.com Date: Wed Nov 16 19:25:20 2016 -0500

```
working on sast
```

commit 76f6c5b517d1fa32a365774783d721fea7600179 Author: alan97 duanyanl97@gmail.com Date: Wed Nov 16 18:08:00 2016 -0500

```
Fixed bug in rusty; Finished binop in codegen
```

commit aec44aab5bbd58cc25def158bf786029e05d48b6 Author: alan97 duanyanl97@gmail.com Date: Wed Nov 16 17:57:15 2016 -0500

```
fixed semant.ml syntax error
```

commit 2fb5d0e0b28da2a1f97898212fff4f3902fc9ebe Merge: 615b38d 2c3cc7b Author: alan97 duanyanl97@gmail.com Date: Wed Nov 16 17:43:39 2016 -0500

```
changed code
```

commit 615b38d05125af5a3f49d6475a2456341a4dd827 Author: alan97 duanyanl97@gmail.com Date: Wed Nov 16 17:42:49 2016 -0500

```
changed codegen
```

commit 2c3cc7b8ce4b93888cca7d28cbb852ea0faaab72 Author: Sherry Qiu sherryq6@gmail.com Date: Wed Nov 16 00:23:38 2016 -0500

```
fix syntax codegen
```

```
commit 58da825e4fd6034e3142039cd39331bb38c8eab7 Author: Emily Meng ewm2136@columbia.edu Date:
Mon Nov 14 01:53:26 2016 -0500

    finished loops

commit a6986b8d9bcc79f573d6e07bce2c4399ad7145e1 Merge: f17e042 c74ff27 Author: zk2202 zk2202@
columbia.edu Date: Mon Nov 14 00:46:06 2016 -0500

    Merge branch 'master' of https://bitbucket.org/sherryq/rusty

commit f17e0422bc9e767e0a867532bb51306ed19d0872 Author: zk2202 zk2202@columbia.edu Date: Mon Nov
14 00:41:06 2016 -0500

    testcases about arithmethic and few changes about exception messages in
        semant.ml

commit c74ff27bec41e856cec83b2375d9827309097ed0 Author: Emily Meng ewm2136@columbia.edu Date:
Sun Nov 13 22:30:53 2016 -0500

    if and while done for and loop in progress

commit 91b8dd8ab61abe389b6f5a4c8317036f9616528b Author: Emily Meng ewm2136@columbia.edu Date:
Sun Nov 13 20:20:28 2016 -0500

    return for type and void

commit b2e2b013786f7c2283786a5927aa90c35587eef1 Author: Emily Meng ewm2136@columbia.edu Date:
Sun Nov 13 20:01:55 2016 -0500

    layout for control flow

commit 72877d9d157a5854498ced66d6b3cb75c921b527 Author: alan97 duanyanl97@gmail.com Date: Sun
Nov 13 14:53:07 2016 -0500

    Rewrote semant.ml; now we output sast

commit 6a6d86695cd1c034811dcfc3f72480919f31e8f2 Author: yanlind yanlind@andrew.cmu.edu Date: Mon
Nov 7 02:26:13 2016 -0500

    Now full pipeline working! Today rusty says \'Hello world\!\' to everyone\!

commit 251b7b29445f4fb32cbb99f9972902d9d1f86ef8 Author: zk2202 zk2202@columbia.edu Date: Sun Nov
6 20:39:05 2016 -0500

    Finished makefile and rusty.ml, waiting for syntax error debugging...

commit 288161c28b7b24a435883c9089d37f1763c839f6 Author: yanlind yanlind@andrew.cmu.edu Date: Sun
Nov 6 20:16:45 2016 -0500

    deleted the ocaml generated files

commit d001249efab31b335def7dfb55bf27f5bdcdf0a6 Author: yanlind yanlind@andrew.cmu.edu Date: Sun
Nov 6 20:14:50 2016 -0500

    fixed little bugs in codegen and ast

commit f1d07734029520aa9e7356edb927ef1d33b1b295 Author: yanlind yanlind@andrew.cmu.edu Date: Sun
Nov 6 20:02:24 2016 -0500

    rusty.ml added
```

commit fe037cdcb827cfaf4e626355224503d3fc28dfa5 Author: yanlind yanlind@andrew.cmu.edu Date: Sun Nov 6 19:57:49 2016 -0500

```
Codegen.ml updated with printf handles either int or string input
```

commit b10688ca02a8ccc7731634ca930e3e14a7d304b7 Author: Emily Meng ewm2136@columbia.edu Date: Sun Nov 6 18:55:47 2016 -0500

```
string
```

commit 459800e4a537640f2c2fc148dbd100fd8d05218b Author: Sherry Qiu sherryq6@gmail.com Date: Sun Nov 6 18:40:12 2016 -0500

```
rm printb
```

commit bef2f05dbc4fa9dfc9e93634a438e643bf2159a3 Author: Sherry Qiu sherryq6@gmail.com Date: Sun Nov 6 18:39:07 2016 -0500

```
change printf to println
```

commit df74ce7b6f2fe0e068eec4c40026f67c99d0cb57 Author: Emily Meng ewm2136@columbia.edu Date: Sun Nov 6 18:33:16 2016 -0500

```
comment out non hello world
```

commit 832a7fa9901bb474718d6331860d9379907a6a28 Author: Emily Meng ewm2136@columbia.edu Date: Sun Nov 6 18:27:12 2016 -0500

```
moved codegen
```

commit 617a366e842a7a339d1524f0fb1c08fa3bdeebef Author: Emily Meng ewm2136@columbia.edu Date: Sun Nov 6 18:24:08 2016 -0500

```
initial changes to match AST
```

commit a1979dff35158731efddf09a2192c74313c87457 Author: yanlind yanlind@andrew.cmu.edu Date: Sun Nov 6 18:22:19 2016 -0500

```
Updated semanic check
```

commit d8f1be0c5483b26ef9241912354b1747610c9d5c Author: yanlind yanlind@andrew.cmu.edu Date: Sun Nov 6 18:17:18 2016 -0500

```
updated semantic check
```

commit ea36e36af5da8ed74d87d43b9db0f81035135405 Author: Emily Meng ewm2136@columbia.edu Date: Sun Nov 6 18:00:09 2016 -0500

```
comment out global variables
```

commit 7d2ad856d9ac523e3a3f773146355b4daada8b2b Author: yanlind yanlind@andrew.cmu.edu Date: Sun Nov 6 17:57:57 2016 -0500

```
Added string to ast, parser
```

commit e7825995fac3119bfaf9a00616749414c12fcc52 Author: Sherry Qiu sherryq6@gmail.com Date: Sun Nov 6 17:47:50 2016 -0500

```
check args for main fn
```

commit f8d2548ce4fbfcfa2f8a24c81a8c0d52492e16d2 Author: Sherry Qiu sherryq6@gmail.com Date: Sun Nov 6 16:38:19 2016 -0500

```
add helper functions
```

commit 303c139c9c8ec2f8149002aed01918eeda295c5c Author: Sherry Qiu sherryq6@gmail.com Date: Sun Nov 6 16:35:05 2016 -0500

```
init semant.ml
```

commit e71ca8e66eaadd6da86c4f93fc9e5ce9708760a2 Author: yanlind yanlind@andrew.cmu.edu Date: Wed Oct 26 20:57:10 2016 -0400

```
struct method call added as expr
```

commit 6692aa6bfb4f78055d04600ce0ce199058f9f2e1 Author: yanlind yanlind@andrew.cmu.edu Date: Wed Oct 26 20:50:43 2016 -0400

```
struct impl added
```

commit 7b3e07951012ae83897b14548e66e11511be027f Author: yanlind yanlind@andrew.cmu.edu Date: Wed Oct 26 20:47:27 2016 -0400

```
added unary +
```

commit a49d7156b90c3a31f3811db9694e35f35be6dc78 Author: yanlind yanlind@andrew.cmu.edu Date: Wed Oct 26 15:07:54 2016 -0400

```
added void type
```

commit 3e550dac215d272e7dbebb50ad7416e5dd42f8b3 Author: yanlind yanlind@andrew.cmu.edu Date: Sun Oct 23 20:19:55 2016 -0400

```
Added comments in parser
```

commit e1ec8f6adc1b6c319eeae540597b15e3efb06267 Author: yanlind yanlind@andrew.cmu.edu Date: Sun Oct 23 18:33:12 2016 -0400

```
No shift/reduce conflict
```

commit e1416b2cba64aa9a44cee7c24899a812d853241d Author: yanlind yanlind@andrew.cmu.edu Date: Sun Oct 23 02:07:28 2016 -0400

```
Finished scanner; ast needs to add memory safety; parser unchanged
```

commit d31eff1c4ae52d429f5406a1f66205e8b253da8b Author: yanlind yanlind@andrew.cmu.edu Date: Sat Oct 22 17:10:16 2016 -0400

```
Go back to original MicroC
```

commit 43e60acb5c9b37306e5222e9bbbe387a55fdb976 Author: yanlind yanlind@andrew.cmu.edu Date: Sat Oct 22 16:51:38 2016 -0400

```
also added MicroC test cases
```

commit 8bfd808d13c26ffb639f4a08b96b325fb8232c7d Author: yanlind yanlind@andrew.cmu.edu Date: Sat Oct 22 16:31:30 2016 -0400

```
Initial Version of scanner, parse and ast (based on MicroC)
```

commit a490e947011b2367319269e737d793b50858946b Author: yanlind yanlind@andrew.cmu.edu Date: Sat Oct 8 22:01:55 2016 -0400

```
added lrm, ref and readme - yd
```

commit 169f8eedda233b1d2014e7b55c3d95ee5917b12c Author: yanlind yanlind@andrew.cmu.edu Date: Sat Oct 8 20:21:48 2016 -0400

```
Initial commit with contributors: Yanlin Duan
```

Compiler Architecture

The rusty compiler consists of 6 main components:

- **scanner.mll** : scanner takes in the .rusty file and uses predefined keywords to match and break the file into tokens.
- Input: .rusty file
- Output: tokens
- Errors caught at this stage: Illegal characters
- **parser.mly** : Uses ocamlyacc to write a parser that parses the keywords and generates an abstract syntax tree.
- Input: tokens matched in scanner
- Output: AST
- Errors caught at this stage: Syntax errors
- **pre-semant.ml** : pre-semant takes in the AST that the parser generated and does de-sugaring. The major changes in this stage are:

1) extracting struct impl (struct methods), deleting those nodes (to a NoExpr), and transforming them to a func_decl. The function name will be a concatenation of struct name and impl name. We will also pass in "self" as the first argument in the function, which is of struct type.

2) changing all structMethodCall to a function Call, and pass in the struct as the argument.

Say we have a struct variable called origin, with type Point, and Point has a impl called calculateDistance(). Then origin.calculateDistance() will become a function call: PointcalculateDistance(origin).

- Input: AST from parser
- Output: de-sugared AST
- Errors caught at this stage: impl associated with undefined struct, call an undefined impl, etc.
- **semant.ml** : Performs static check on AST and, if all tests are passed, generate semantically-checked AST (SAST).

The semant will first create some built-in functions' signature (for example, println), and then add them into a function_decls string map. It will then add in all user defined functions (and previously transformed struct impls) to the same map.

During this process, it will report any duplicated functions, any overwriting of built-in functions and make sure main function is properly defined.

It will then start to convert all fdecl (function declarations) to sfdecl (semantically-checked function declarations, with types associated to each node). This involves recursively calling stmt_to_sstmt as well as expr_to_sexpr.

The type checking includes, but is not limited to: undeclared variable name; void variable type; if array contains multiple types; struct access is valid; binary operators (including variable binding and assignment) and unary operators are operating on correct and consistent types and everything makes sense; function signature does not have duplicated arguments; amounts and types of function formals and actuals match; function return type and real return value type match; predicates in if, while, for evaluate to boolean.

In semant we also implemented zero-cost abstractions to help prevent data race and segfaults. In particular we have two systems, ownership system and borrow system, which will be introduced in detail here.

The ownership and borrow checking includes, but is not limited to: use of moved value; mutable borrow of immutable variable; change of immutable value; violation of borrow rules (discussed in detail in Memory Safety section); dereferencing a non-borrow.

During the check, semant.ml will also maintain an environment variable containing variable bindings in current scope. Leveraging on the functional programming's features in OCaml, we can easily maintain different environments in different scopes by creating a new env that will only be locally visible. So this will not overwrite the env in outer layer. By doing this, variable shadowing (defining same name variable in different scopes without collision) can also be achieved.

All nodes in the AST will change to SAST nodes during the checking. The purpose of the SAST is to help codegen to generate LLVM code by passing in more information and stopping code from going to next steps if any errors have been found.

- Input: AST from pre-semant
- Output: SAST with type annotations
- Errors caught at this stage: type errors, undeclared value errors, breaking the ownership & borrow systems, etc.
- **codegen.ml** : codegen.ml utilizes OCaml LLVM bindings to compile rusty code to LLVM code. It takes in the SAST from semant.ml, creates the context, module, necessary types and builder, and does book-keeping on different basic blocks.

The codegen will also maintain environment (an OCaml record), as well as some variable hashtable, struct hashtable and function declaration hash table. Similar to semant, codegen will walk through the whole tree and spit out LLVM values as needed.

We will highlight some of the smart tricks in our codegen here:

binding with C function : We use L.declare_function to bind our println function with C's printf function. Using pattern matching, we resolve the type of the argument and achieve an Ad-hoc polymorphism in this case.

Use L.struct_type to define lltype for struct : Struct may contain heterogeneous types, and they are all user-defined. We have no idea what it may look like and how much memory we should allocate for it. In that regard, we used L.struct_type which allows us to define our own lltype. LLVM will then be able to resolve the size and allocate / deallocate / access memory as usual.

A two-pass generation : rusty does not require struct definitions above all function definitions. Rather, they can (and should) be defined within functions and they will be visible everywhere. A problem with that would be, when codegen passes through all functions, it may see a function that takes in a struct type, but it has no idea what the type is and what size it should be.

To achieve this, we did a two-pass in codegen where in the first pass, codegen will only look at struct definitions and put them in a struct map, where key is the struct name and value is the corresponding lltype. In the second-pass, codegen should have seen all the structs defined (otherwise the error will be caught in semant) and it can declare the function_type as usual.

Heap-allocated data structure: array and struct : To give more flexibility to programmers as well as to facilitate our memory safety guarantee, we chose to allocate arrays and structs on the heap. This will

involve a totally different set of LLVM functions for allocating on the heap, deallocating on the heap, pointer casts, pointer arithmetic, etc, but we managed to make them work coherently.

A preliminary version of reference counting : We have a very naive attempt at reference counting in rusty. In Rust (the language that inspired rusty), memory deallocation happens deterministically when variables fall out of scope and the resources they own are not used by any other variables. We tried to do a similar thing, that we check at the end of the scope whether any resources on the heap can be freed. We will only free whatever we are sure won't be needed. This code currently only operates on arrays on a very limited scale.

- Input: SAST
- Output: LLVM IR
- Errors caught at this stage: Minimal error checking as most problems should be identified in the previous steps.
- **rusty.ml** : Finally we have rusty.ml, which is a short-and-sweet file that serves both as the front-end of our compiler (that generates the rusty.native) and as a pipeline that connects everything above.
- Input: .rusty file
- Output: LLVM IR
- Errors caught at this stage: N/A

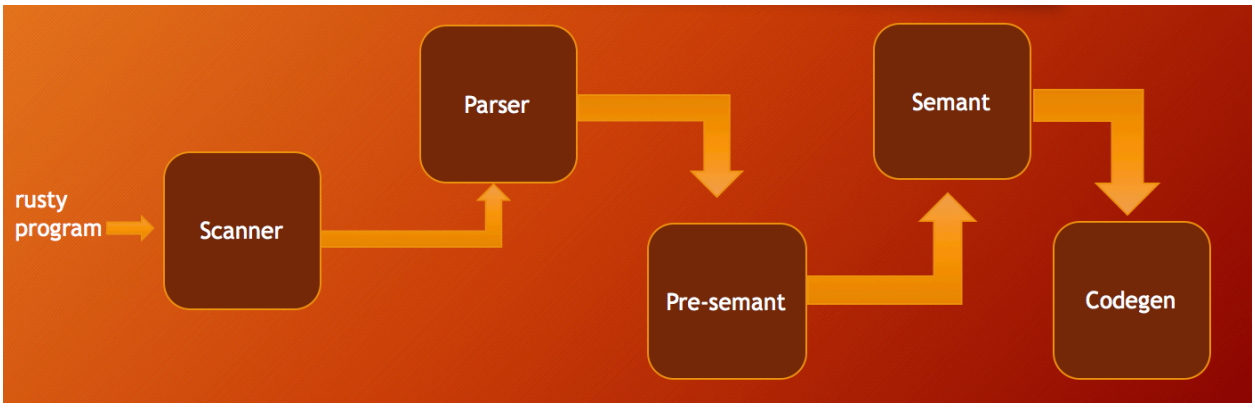

Figure 2: Block diagram of major components of rusty compiler

Utilities

rusty compiler will not come into existence without our utilities. They are not part of the compiler pipeline *per se*, but they either define an important data structure we use, or help us make the compiler functional well.

- **ast.ml**: ast.ml defines the abstract syntax tree, as well as primitive and composite types. It also consists of some pretty print functions including string_of_typ, string_of_expr that are frequently used when debugging.
- **sast.ml**: sast.ml defines the semantically-checked abstract syntax tree. In short it is a type-annotated version of ast. As in ast.ml, here we also put in utility functions such as get_type_from_sexpr and string_of_sexpr.
- **makefile**: we use Makefile to build our compiler.
- **testall.sh**: we wrote a bash script to run our regression test suite on our compiler.

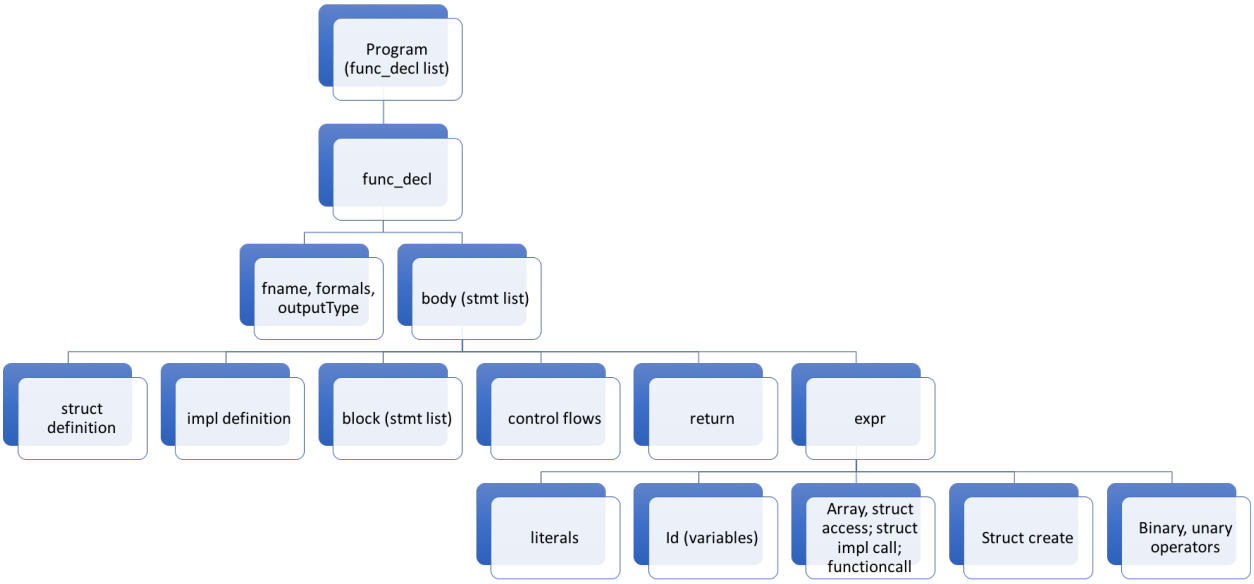

Figure 3: A graphical representation of AST

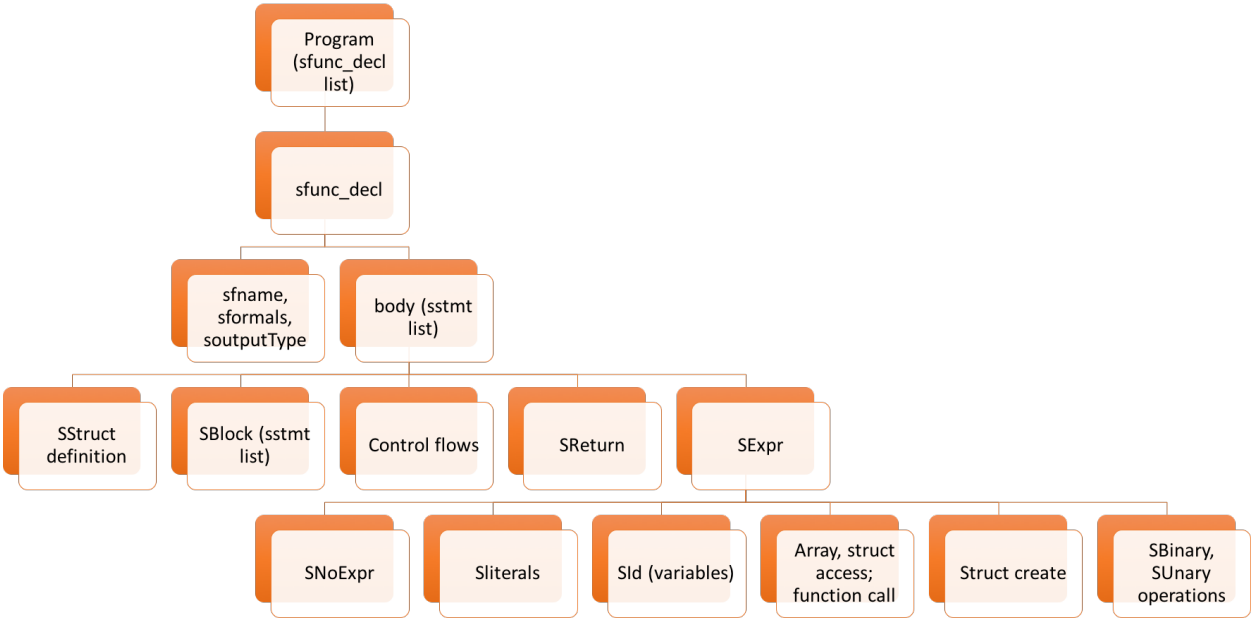

Figure 4: A graphical representation of SAST

- **git, bitbucket and gitbook**: we use git for version control, and bitbucket to store our code. We use gitbook to document and publish our LRM, architectual design, language tutorials, etc.

Responsibilities

Although we all contributed to each component in the architecture at some point during this project, the main responsibilities were divided as follows:

Member	Component
Yanlin Duan	Scanner, parser, ast, sast, semant, codegen, rusty.ml
Zhuo Kong	rusty.ml, ast, codegen
Emily Meng	Parser, ast, sast, pre-semant, semant
Shiyu Qiu	Parser, ast, sast, codegen

How rusty helps prevent segfault and data races

rusty, inspired by Rust, takes memory safety as its primary attempt. We hate data race and segment fault, as many, if not most, programmers. rusty achieves data safety goal through 'zero-cost abstractions', which means that in rusty, abstractions cost as little as possible in order to make them work.

One of our attempts toward memory safety is called the ownership system. All of the analysis are *done at compile time* (i.e. in semant.ml), so programmers do not pay any run-time cost for any of these features.

Ownership system

In rusty, variable binding means that a certain variable owns a resource, and we ensure that there is *exactly one* binding to any given resource.

For example, if you do:

```
fn main () -> int {

  let v: [int;5] = [1,2,3,4,5];
  let mut v2: [int;5] = v;
  println(v[0]);
  return 0;

}
```

This will throw an error with the message "use of moved value v", which is triggered by line 5. This is because that in line 4: we transfer ownership to v2 , and that's what we meant by saying that we have moved that array and v no longer owns it. When we try to access v[0], it will error out.

Why do we do this?

When we move v to v2, rusty copies the pointer to an array (allocated on heap), v, into the stack allocation represented by v2. This shallow copy does not create a copy of the heap allocation containing the actual data. Therefore, there are two pointers pointing to the same area in the heap.

Now you may already know why this is bad: if v2 changes the array, say truncates some elements in it, then v and v2 will disagree on the length of the array and therefore we may access to memory we are not supposed to. This is why rusty forbids using v after the move.

Immutable, mutable borrowing and borrow checker

As you can see, it will be very tedious to keep handing ownership back and forth, especially when there are function calls. Instead, there is a short-cut: instead of moving resources, we "borrow" them.

A borrow is like a reference. For example:

```
fn main () -> int {

  let v: int = 42;
  let v2: &int = &v;
  println(*v2); /* 42 */
  println(v); /* 42 */
  return 0;

}
```

This is perfectly valid. We are no longer moving the resource to v2. Rather v2 is just borrowing it. Also note that the borrowing is an immutable one, that is, if v2 tries to dereference and change the value, compiler will complain.

Here is a quick demo of mutable borrow:

```
fn main () -> int {

  let mut v: int = 42;
  let mut v2: &mut int = &mut v;
  *v2 = *v2 + 1;
  println(*v2); /* 43 */
  return 0;

}
```

As you can see, we deliberately make the mutable borrow very verbose, as programers should be very cautious about this. Line 4 basically reads "let's define a mutable variable v2, which is of type a mutable reference to int, and bind the mutable borrow of v to it".

You may wonder why we are not printing the value of v. This is because in this case, it is not permitted. when we try to print v, or access v, the println function essentially immutably borrows v and tries to read the value. However, when you can read (through v) and write (through v2) at the same time, bad things happen.

Here are the rules for borrowing:

You may have one or the other of these two kinds of borrows, but not both at the same time:

- one or more references (&T) to a resource
- exactly one mutable reference (&mut T)

Recall the definition of data race:

"There is a 'data race' when two or more pointers access the same memory location at the same time, where **at least one of them is writing**, and the operations are not synchronized"

You can tell how the borrowing rule in rusty is matching with the definition of data race and tries to prevent it from happening.

An naive 'reference counting' scheme

rusty collects (frees) memory deterministically at the end of the scope. We implemented a very naive version of reference counting, where we will free all memory that we are sure no one owns. This scheme only works on 1-dimensional array as of now, So for example:

```
fn main () -> int {

  let v: [int;5] = [1,2,3,4,5];
  {

    let v: [int;5] = [6,7,8,9,10]; /* can have the same name (shadowing) */

  } /* v gets collected and freed */

  println(v[3]); /* 4 */
  return 0;

} /* the outer v gets collected and freed */
```

What to do next

We believe our anti-data-racing scheme is very promising (and that is why Rust is becoming more and more popular right now!). Rust also implements lifetime rule, which further facilities the borrow checker and covers more edge cases. If we have more time, we will be working on lifetime system and make the check more rigorous.

Test Plan

We wrote unit tests for features originally outlined in our LRM and added additional tests as we developed, testing to fail to check for exception results. Every test case has a specific name indicating the feature or error it tested and can be easily identified from the console. All tests execute automatically by running a shell script and output is piped to a log file, where we can see more details about which tests failed and where they failed. As we developed our compiler, the tests were sorted into different folders under the main test folder, categorizing the tests based on whether or not the feature has been implemented yet. We mainly tested by writing source code with println statements and comparing the output of the generated code with an expected output.

Source to Target Examples

- Hello World

test_helloworld.rusty

```
fn main() -> int {
    println("Hello world!");
    return 0;
}
```

test_helloworld.ll

```
ModuleID = 'rusty'
@imt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt = private unnamed_addr constant [7 x i8] c"%0.2f\0A\00"
@cmt = private unnamed_addr constant [4 x i8] c"%e\0A\00"
@sfmt = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@printstr = private unnamed_addr constant [14 x i8] c"Hello world!\0A\00"
declare i32 @printf(i8*, ...)
define i32 @main() {
entry:
%printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([14 x i8],
    [14 x i8]* @printstr, i32 0, i32 0))
ret i32 0
}
```

test_helloworld.out

```
Hello world!
```

- Struct

test_struct4.rusty

```
fn main() -> int{
    struct Point {
        x : float,
        y : float
    }
    let origin : Point = { x : 42.1, y : 1.1 };
    println(origin.x + origin.y);
}
```

test_struct4.ll

```
; ModuleID = 'rusty'
@imt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt = private unnamed_addr constant [7 x i8] c"%0.2f\0A\00"
@cmt = private unnamed_addr constant [4 x i8] c"%e\0A\00"
@sfmt = private unnamed_addr constant [4 x i8] c"%s\0A\00"
declare i32 @printf(i8*, ...)
define i32 @main() {
entry:
%malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint
    (double* getelementptr
(double, double* null, i32 1) to i64), i64 2) to i32))
%tmp = bitcast i8* %malloccall to { double, double }*
%tmp1 = getelementptr inbounds { double, double }, { double, double }* %tmp,
    i32 0, i32 0
store double 4.210000e+01, double* %tmp1
%tmp2 = getelementptr inbounds { double, double }, { double, double }* %tmp,
    i32 0, i32 1
store double 1.100000e+00, double* %tmp2
%origin = alloca { double, double }*
store { double, double }* %tmp, { double, double }** %origin
%tmp3 = load { double, double }*, { double, double }** %origin
%tmp4 = getelementptr inbounds { double, double }, { double, double }* %tmp3,
    i32 0, i32 0
```

```
%tmp5 = load double, double* %tmp4
%tmp6 = load { double, double }*, { double, double }** %origin
%tmp7 = getelementptr inbounds { double, double }, { double, double }* %tmp6,
    i32 0, i32 1
%tmp8 = load double, double* %tmp7
%tmp9 = fadd double %tmp5, %tmp8
%printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([7 x i8],
    [7 x i8]* @fmt, i32 0, i32 0),
double %tmp9)
ret i32 0
}
declare noalias i8* @malloc(i32)
```

test_struct4.out

```
43.20
```

Memory Safety

test_functakepointer.rusty

```
fn add ( x: &mut int, y: int )-> void{
    *x = *x+y;
    println(*x);
}
fn main() -> int {
    let mut x : int =4;
    let mut y : &mut int = &mut x;
    add (y,3);
    return 0;
}
```

test_functakepointer.ll

```
; ModuleID = 'rusty'

@imt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt = private unnamed_addr constant [7 x i8] c"%0.2f\0A\00"
@cmt = private unnamed_addr constant [4 x i8] c"%e\0A\00"
@sfmt = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@imt.1 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.2 = private unnamed_addr constant [7 x i8] c"%0.2f\0A\00"
@cmt.3 = private unnamed_addr constant [4 x i8] c"%e\0A\00"
@sfmt.4 = private unnamed_addr constant [4 x i8] c"%s\0A\00"

declare i32 @printf(i8*, ...)

define i32 @main() {
entry:
  %x = alloca i32
  store i32 4, i32* %x
  %x1 = load i32, i32* %x
  %y = alloca i32*
  store i32* %x, i32** %y
  %y2 = load i32*, i32** %y
  call void @add(i32* %y2, i32 3)
```

```
  ret i32 0
}

define void @add(i32*, i32) {
entry:
  %x = alloca i32*
  store i32* %0, i32** %x
  %y = alloca i32
  store i32 %1, i32* %y
  %x1 = load i32*, i32** %x
  %tmp = load i32, i32* %x1
  %x2 = load i32*, i32** %x
  %tmp3 = load i32, i32* %x2
  %y4 = load i32, i32* %y
  %addtmp = add i32 %tmp3, %y4
  %x5 = load i32*, i32** %x
  store i32 %addtmp, i32* %x5
  %x6 = load i32*, i32** %x
  %tmp7 = load i32, i32* %x6
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8],
      [4 x i8]* @imt.1, i32 0, i32 0),
  i32 %tmp7)
  ret void
}
```

test_functakepointer.out

```
7
```

Test cases

Our test suite includes 86 test cases that cover all of the different features we have in our language including arithmetic operations, control flow, arrays, structs, and memory safety. The tests were categorized based on what set of features they tested.

- Function (includes types and variables)

These tests test functions. We check for duplicate of function names and variable names, correct function input types and return types, functions calling other functions, and passing values into functions and validating those results.

- Operators (binary and unary operator)

These test cases were chosen because they check whether or not all of the operators are working properly. In addition to operator functionality checking, the tests were written to include type checking, operands of different types, as well as variable binding and arithmetic rules.

- Control Flow

These tests were made to test for if/else blocks, while loops, and for loops. Some of the tests were included in the earlier sections for function and arithmetic testing. These tests aim to check if the control flow is working properly, while still type checking.

- Arrays

These tests make sure arrays have correct syntax, can be declared with all kinds of types, and are used correctly.

Figure 5: Functions

```
justwe@justwe137:~/workspace/addsast/rusty$ ./testall.sh
test_arith1_add... OK
test_arith2_sub... OK
test_arith3_mult... OK
test_arith4_division... OK
test_arith5_mod... OK
test_arith6_equal... OK
test_arith7_noteq... OK
test_arith8_relation... OK
test_arith9_andornot... OK
fail_arith11_or... OK
fail_arith12_not... OK
fail_arith13_less... OK
fail_arith14_leq... OK
fail_arith15_greater... OK
fail_arith16_geq... OK
fail_arith1_division... OK
fail_arith4_and... OK
fail_arith9_modzero... OK
justwe@justwe137:~/workspace/addsast/rusty$
```

Figure 6: Operators

```
justwe@justwe137:~/workspace/addsast/rusty$ ./testall.sh
test_boolreturn... OK
test_for... OK
test_for2... OK
test_if... OK
test_if3... OK
test_while1... OK
test_while2... OK
fail_for1... OK
justwe@justwe137:~/workspace/addsast/rusty$
```

Figure 7: Control Flow

```
justwe@justwe137:~/workspace/addsast/rusty$ ./testall.sh
test_array1... OK
test_array2... OK
test_arraystring... OK
```

Figure 8: Arrays

- Structs

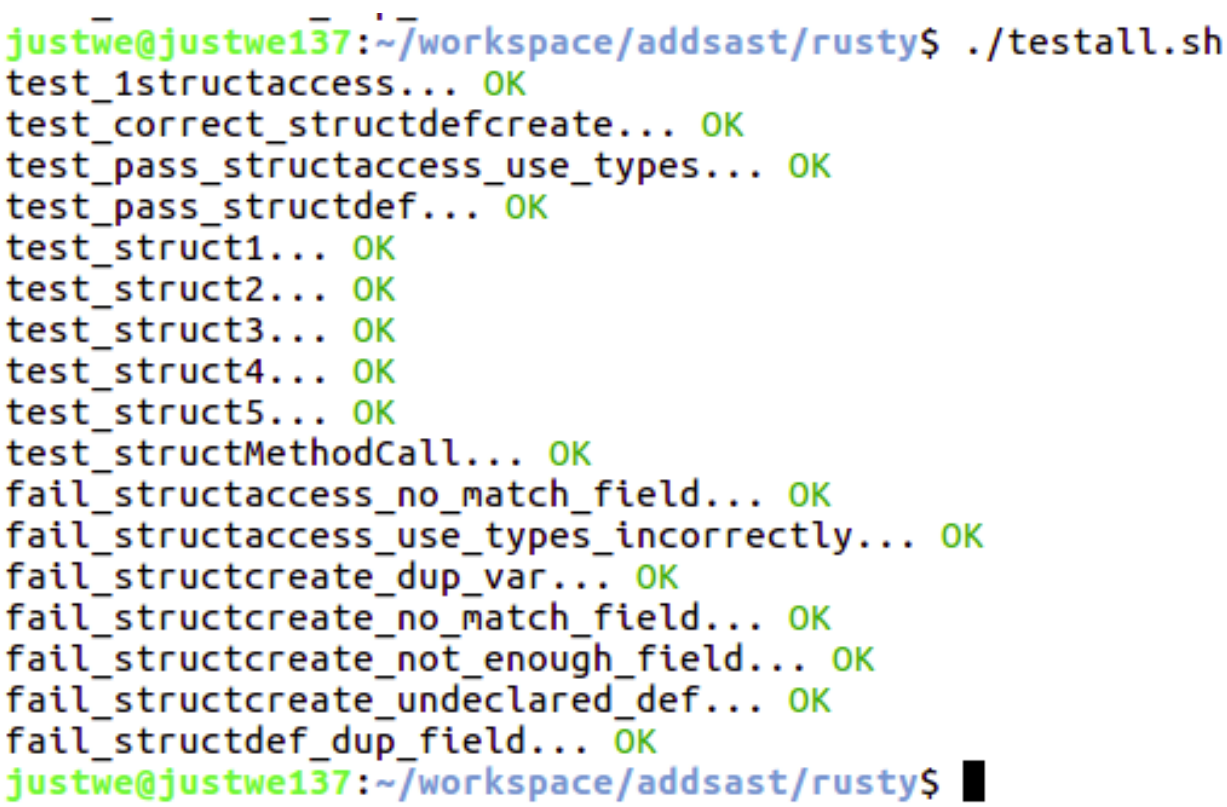

Figure 9: Structs

These tests were used to test if structs functioned exactly as we intended them to. There are tests for creating and accessing structs, calling methods associated with structs, and passing values of structs into functions. In addition, we have tests that check for raising the correct exceptions regarding structs, such as duplicate field names or type mismatch between struct definition and declaration.

- Memory Safety

These tests were used to ensure our language correctly followed the rules we designed for ownership and borrowing to ensure memory safety. We test various combinations of mutable and immutable variables with the different operators for borrowing and referencing.

Automation

Testing was automated with a test script. Test all the test cases in the test folder by running the command:

```
./testall.sh
```

Responsibilities

Member	Component
Yanlin Duan	Memory safety

Member	Component
Zhuo Kong	Test script, operators
Emily Meng	Structs
Shiyu Qiu	Functions, control flow, arrays, struct

Yanlin (Alan) Duan

I learned two important lessons during this project.

First is the importance of **building your debugging tools**. For example, it is very common for us to raise a failure and output the string of type of an llvalue. If we write it every time we need it, it is a pain and it diverts us from solving the real problem (just seeing what is going on is painful enough!). Next time if I do similar projects, I will definitely devote more time in building helpful debugging functions (even modules), the return of which definitely overweighs the cost.

Second is **taking ownership, and acting as a team-player**.

I think PLT teaches me to take ownership. **I own what I wrote, so when there is trouble, I am also responsible to resolve the issue. It is not okay for me to just sit there and say "I don't know what's going on" and let it be.** It forces me to debug in the right way: find the minimum test cases that reproduce the error, read assembly language, search on stackoverflow, set breakpoint, output useful information, learn OCaml, know LLVM well..., and when the project ends, I grow the most.

At the same time, PLT is not an individual project. We have team mates and we ought to act as a team. I also learned to be a good team player during this project. In the past I was not very comfortable (or sometimes because of laziness) to explain what I thought or what was going on to my teammates, but here in PLT, I have to effectively communicate to other people about what I want to achieve, how I achieve it and why, otherwise, similar problems would occur again and again.

Also, PLT gives me the chance to become the go-to person in our team. Though I am officially the system architect, I actually involve actively in almost every step of our compiler work– either through writing the source code, or by helping tracing down some troublesome issues. When other people brought issues to me, I am happy to help. I enjoy this because it not only helps me to grow faster and get exposed to more things, but also help the team get stronger.

Shiyu Qiu

As team manager, the most important lesson for me this semester was how to communicate effectively and work together as a team. After many years of doing assignments as an individual or working on projects with friends, it's difficult to collaborate well with people you've only just met. It's crucial to keep in mind that everyone comes from different backgrounds. We've all learned our own way of problem-solving and collaborating with others, and ideas that we take for granted are not always what come to mind for others. We need to have patience when we run into problems as a group, and we need to do our best to understand each other. Before this semester, I thought that for group projects we all just needed to divide our work and make sure we each complete our part. However, after this semester I've realized that there is a need for a manager within a group – someone to make sure that everyone is on the same page and to foresee conflicts before they occur.

In terms of programming, I've learned a lot about functional programming and to logically think through my code. With other languages that I'm more familiar with, such as Java and Python, I don't have to think through my program as much since the compiler will tell me exactly what the error was. However, with fewer specific messages from Ocaml and LLVM, it's important to understand my code logically to effectively and efficiently contribute to the project.

```
justwe@justwe137:~/workspace/addsast/rusty$ ./testall.sh
test_1borrowmix... OK
test_1derefenceNonBorrow... OK
test_1immutBorrowMut... OK
test_functakepoint... OK
test_mut1... OK
fail_1borrowmix... OK
fail_1derefenceNonBorrow... OK
fail_1letmutimut... OK
fail_1mutBorrowImmut... OK
fail_1usemovedvalue... OK
fail_2borrowmix... OK
fail_2changeimmutborrow... OK
fail_2letimmutmut... OK
fail_2usemovedvalue... OK
fail_3changeImmutVar... OK
fail_3usemovedvalue... OK
fail_mut11... OK
justwe@justwe137:~/workspace/addsast/rusty$
```

Figure 10: Memory Safety

Emily Meng

As Professor Edwards states at the beginning of class, the hardest part of the project isn't necessarily the programming part, but being able to work efficiently with your team. OCaml and the functional programming style was really hard to wrap my head around, especially since you go through a fairly long period of time where you barely use it between the first homework assignment and when you can actually start to program more substantial parts of your compiler. You end up getting used to it after staring at it for a couple of hours though. When designing and assembling your compiler, don't follow MicroC too closely. It works as it is since it is very simple, but building a language with more features requires a decent amount of planning to make sure you aren't hacking things together at the end to make them work.

Zhuo Kong

For testing and debugging, I learned two things. First, always check if the previously passing test cases are still working and make sure every test case is designed as I intended it to be. Each test case should indicate only one exception. For debugging, the most important thing is to write exception messages while coding, so I know that if the test case has failed, I can find where the problem is. Also, I have learned a lot about how to write tricky test cases for each feature and understanding where features could go wrong in the language, so I can write tests to examine them. For teamwork I must give great thanks to my three teammates. We have a really complicated language structure and as a tester, I needed to ask a lot of questions about the program and sometimes let other members handle debugging the failures. It is very important to communicate with others and always be engaged with the project.

Advice for future teams

- **Start early!! Have a schedule and stick to it!!**
- Learn OCaml, LLVM and OCaml - LLVM binding well!
- Set up standards (code style, naming convention, test case design, etc.) before diving in and coding.
- Have regular meeting time (~4 hours per week at least) and make sure everyone is on the same page.
- Being understanding of each other helps achieve good team work

Appendix

scanner.mll

```
(* signed: Yanlin Duan *)

{
    open Parser
    let lineno = ref 1
    let depth = ref 0
    let filename = ref ""

    let unescape s =
        Scanf.sscanf ("\"" ^ s ^ "\"") "%S%!" (fun x -> x)
}

let alpha = ['a'-'z' 'A'-'Z']
let escape = '\\' ['\\' ''' '"' 'n' 'r' 't']
let escape_char = ''' (escape) '''
let ascii = ([' '-'!' '#'-'[' ']'-'~'])
let digit = ['0'-'9']
let id = alpha (alpha | digit | '_')*
let string = '"' ( (ascii | escape)* as s) '"'
let char = ''' ( ascii | digit ) '''
let float = (digit+) ['.'] digit+
let int = digit+
let whitespace = [' ' '\t' '\r']
let newline = '\n'

rule token = parse
  whitespace { token lexbuf }
| newline      { incr lineno; token lexbuf}
| "/*"     { incr depth; comment lexbuf }
| "//"     { comment2 lexbuf }
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| ';'      { SEMI }
| ':'      { COLON }
| ','      { COMMA }
(* Operators *)
| '+'      { PLUS }
| '-'      { MINUS }
```

```
| '*'       { TIMES }
| '/'       { DIVIDE }
| '%'       { MODULO }
| '='       { ASSIGN }
| "=="      { EQ }
| "!="      { NEQ }
| '<'       { LT }
| "<="      { LEQ }
| ">"       { GT }
| ">="      { GEQ }
| "and"     { AND }
| "or"      { OR }
| "not"     { NOT }
| '.'       { DOT }
| '['       { LBRACK }
| ']'       { RBRACK }
(* Ownership *)
| "&"       { BORROW }
| "mut"     { MUTABLE }
| "&mut"    { MUTABLEBORROW }
(* Branch Control *)
| "if"      { IF }
| "else"    { ELSE }
| "for"     { FOR }
| "while"   { WHILE }
| "loop"    { LOOP }
| "in"      { IN }
(* Data Types *)
| "int"     { INT }
| "float"   { FLOAT }
| "bool"    { BOOL }
| "char"    { CHAR }
| "true"    { TRUE }
| "false"   { FALSE }
| "string"  { STRING }
| "struct"  { STRUCT }
| "impl"    { IMPL }
| "let"     { LET }
| "as"      { AS  }
| "void"    { VOID }
(* function *)
| "fn"      { FUNC }
| "->"      { OUTPUT }
| "return"  { RETURN }

(* Other *)
| int as lxm           { INT_LITERAL(int_of_string lxm) }
| float as lxm         { FLOAT_LITERAL(float_of_string lxm) }
| char as lxm          { CHAR_LITERAL( String.get lxm 1 ) }
| escape_char as lxm{ CHAR_LITERAL( String.get (unescape lxm) 1) }
| string               { STRING_LITERAL(unescape s) }
| id as lxm            { ID(lxm) }
| eof                  { EOF }
| '"'                   { raise (Failure("Unmatched quotation at " ^
```

```
    string_of_int !lineno)) }
| _ as illegal     { raise (Failure("IllegalCharacter: " ^ Char.escaped
    illegal ^ " at " ^ string_of_int !lineno)) }

and comment = parse
    newline { incr lineno; comment lexbuf }
  |    "*/"       { decr depth; if !depth > 0 then comment lexbuf else token
     lexbuf }
  | "/*"       { incr depth; comment lexbuf }
  |    _          { comment lexbuf }

and comment2 = parse
    newline {token lexbuf}
  | _ {comment lexbuf}
```

parser.mly

```
(* signed: Yanlin Duan, Emily Meng, Shiyu Qiu *)

/* Ocamlyacc parser for rusty */

%{
open Ast
%}

%token SEMI LPAREN RPAREN LBRACE RBRACE LBRACK RBRACK COMMA COLON DOT
%token PLUS MINUS TIMES DIVIDE MODULO ASSIGN NOT
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR
%token RETURN IF ELSE FOR IN WHILE LOOP AS INT BOOL FLOAT CHAR STRING
%token BORROW MUTABLE MUTABLEBORROW VOID
%token STRUCT IMPL LET
%token FUNC OUTPUT
%token <int> INT_LITERAL
%token <float> FLOAT_LITERAL
%token <char> CHAR_LITERAL
%token <string> STRING_LITERAL
%token <string> ID
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE MODULO
%right NOT NEG BORROW MUTABLEBORROW DEREF AS
%right RBRACK RPAREN RBRACE
%nonassoc NOACCESS
%left DOT LPAREN LBRACK LBRACE OUTPUT

%start program
```

```
%type <Ast.program> program

%%

program:
  decls EOF { $1 }

decls:
    /* nothing */ { [] }
  | decls fdecl   { $2 :: $1 }

fdecl: /* fn foo(x:int) -> int { ... } */
  FUNC ID LPAREN formals_opt RPAREN OUTPUT typ LBRACE stmt_list RBRACE
  {
     {
      fname = $2;
      formals = $4;
    outputType = $7;
      body = List.rev $9
     }
    }

formals_opt:
    /* nothing */ { [] }
  | formal_list   { List.rev $1 }

typ:
      INT   { DataT(IntT) }
  | BOOL  { DataT(BoolT) }
  | FLOAT { DataT(FloatT) }
    | VOID  { DataT(VoidT) }
  | CHAR  { DataT(CharT) }
  | STRING {StringT}
  | LBRACK typ SEMI ID RBRACK          { ArrayTD($2, $4) }
  | LBRACK typ SEMI INT_LITERAL RBRACK { ArrayT($2, $4) } /* [int;10] type is
    an integer array of length 10 */
  | BORROW typ                         { RefT(Immut, $2) } /* &int */
  | MUTABLEBORROW typ                  { RefT(Mut, $2) } /* &mut int */
  | ID                                 { StructT($1) }

formal_list:
    ID COLON typ                    { [($1, $3)] }
  | formal_list COMMA ID COLON typ { ($3, $5) :: $1 }

struct_list_opt:
  | struct_list { List.rev $1 }

struct_list:
      ID COLON expr                   { [$1, $3] }
  | struct_list COMMA ID COLON expr { ($3, $5) :: $1 }


stmt_list:
  | stmt           { [$1] }
```

```
  | stmt_list stmt { $2 :: $1 }

stmt:
    expr SEMI         { Expr $1 }
  | RETURN SEMI       { Return Noexpr }
  | RETURN expr SEMI { Return $2 }
    | LET ID COLON typ ASSIGN expr SEMI         { Declaration(Immut, ($2,
       $4), $6) } /* Assignment */
  | LET MUTABLE ID COLON typ ASSIGN expr SEMI { Declaration(Mut, ($3, $5),
     $7) }
  | LBRACE stmt_list RBRACE                  { Block(List.rev $2) }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt     { If($3, $5, $7) }
  | FOR LPAREN expr SEMI expr SEMI expr RPAREN stmt { For($3, $5, $7, $9) }
  | LOOP stmt                                { Loop($2) } /* loop {...} is an
     infinite loop */
  | WHILE LPAREN expr RPAREN stmt        { While($3, $5) }
  | STRUCT ID LBRACE formals_opt RBRACE { StructDef($2, $4) } /* Struct */
  | IMPL ID LBRACE fdecl RBRACE          { ImplDef($2, $4) }

expr_list:
      /* nothing */ { [] }
  | expr { [$1] }
  | expr_list COMMA expr { $3 :: $1 }


expr:
    INT_LITERAL       { IntLit($1) } /* Literals */
  | FLOAT_LITERAL     { FloatLit($1) }
  | CHAR_LITERAL      { CharLit($1) }
  | STRING_LITERAL    { StringLit($1) }
  | TRUE              { BoolLit(true) }
  | FALSE             { BoolLit(false) }
  | ID %prec NOACCESS { Id($1) }
  | LBRACK expr_list RBRACK { ArrayLit(List.rev $2) } /* Array */
  | expr LBRACK expr RBRACK { ArrayAccess($1,$3) }
  | LBRACE struct_list_opt RBRACE { StructCreate($2) }
  | expr DOT ID  { StructAccess($1,$3) } /* Point.x is struct access */
  | expr PLUS   expr { Binop($1, Add,   $3) } /* Binary Operation */
  | expr MINUS  expr { Binop($1, Sub,   $3) }
  | expr TIMES  expr { Binop($1, Mult,  $3) }
  | expr DIVIDE expr { Binop($1, Div,   $3) }
  | expr MODULO expr { Binop($1, Mod,   $3) }
  | expr EQ     expr { Binop($1, Equal, $3) }
  | expr NEQ    expr { Binop($1, Neq,   $3) }
  | expr LT     expr { Binop($1, Less,  $3) }
  | expr LEQ    expr { Binop($1, Leq,   $3) }
  | expr GT     expr { Binop($1, Greater, $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3) }
  | expr AND    expr { Binop($1, And,   $3) }
  | expr OR     expr { Binop($1, Or,    $3) }
  | expr AS     typ  { Cast($1, $3) } /* 32 as float */
  | expr ASSIGN  expr  { Binop($1, Assign, $3)}
  | MINUS expr %prec NEG   { Unop(Neg, $2) } /* Unary Operation */
```

```
  | NOT expr               { Unop(Not, $2) }
  | TIMES expr %prec DEREF { Unop(Deref, $2) }
  | BORROW expr            { Unop(Borrow(Immut), $2)}
  | MUTABLEBORROW expr     { Unop(Borrow(Mut), $2)}
  | ID LPAREN actuals_opt RPAREN                { Call($1, $3) } /* Function
    Call */
  | ID COLON COLON ID LPAREN actuals_opt RPAREN { StructMethodCall($1, $4,
    $6) }   /* Call Struct Methods */
  | LPAREN expr RPAREN { $2 } /* Parenthesis */


actuals_opt:
    /* nothing */ { [] }
  | actuals_list  { List.rev $1 }

actuals_list:
    expr                    { [$1] }
  | actuals_list COMMA expr { $3 :: $1 }
```

ast.ml

```
(* signed: Yanlin Duan, Zhuo Kong, Emily Meng, Shiyu Qiu *)

(* Abstract Syntax Tree and functions for printing it *)

type op = Add | Sub | Mult | Div | Mod |
          Equal | Neq | Less | Leq | Greater | Geq |
          And | Or | Assign | AS

type borrowType = Mut | Immut

type uop = Neg | Not | Borrow of borrowType | Deref

type primitive = IntT | BoolT | FloatT | CharT | VoidT

type typ = DataT of primitive |
                    StringT |
           StructT of string |
                    RefT of borrowType * typ |
                    ArrayT of typ * int|
           ArrayTD of typ * string

and bind = string * typ

and variable_decl = borrowType * bind

and expr =
  | Noexpr
  | IntLit of int
  | BoolLit of bool
  | FloatLit of float
  | CharLit of char
  | ArrayLit of expr list
  | StringLit of string
```

```
  | ArrayAccess of expr * expr
  | StructCreate of ((string * expr) list)
  | StructAccess of expr * string
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Call of string * expr list
    | Cast of expr * typ
  | StructMethodCall of string * string * expr list

type stmt =
    Block of stmt list
    | Declaration of borrowType * bind * expr
    | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt
  | Loop of stmt
  | StructDef of string * bind list
  | ImplDef of string * func_decl
  | Break

and func_decl = {
    fname : string;
    formals : bind list;
    outputType : typ;
    body : stmt list;
}

type program = func_decl list

(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Mod -> "%"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "and"
  | Or -> "or"
    | Assign -> "="
  | AS -> "AS"

let string_of_uop = function
    Neg -> "-"
  | Not -> "!"
```

```
  | Deref -> "*"
  | Borrow(Immut) -> "&"
  | Borrow(Mut) -> "&mut"

let rec string_of_expr = function
    StringLit(s) -> s
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Noexpr -> ""
  | IntLit(n) -> string_of_int n
  | BoolLit(b) -> string_of_bool b
  | FloatLit(f) -> string_of_float f
  | CharLit(c) -> Char.escaped c
  | _ -> "string_of_expr not implemented yet"

let string_of_primitive = function
    IntT -> "int"
  | BoolT -> "bool"
  | FloatT -> "float"
  | CharT -> "char"
    | VoidT -> "void"

let rec string_of_typ = function
    DataT(t) -> string_of_primitive t
  | ArrayT(t,i) -> "["^string_of_typ t^";"^string_of_int  i^"]"
  | ArrayTD(t,s) -> "["^string_of_typ t^";"^ s ^"]"
  | StructT(t) -> t
  | StringT -> "string"
  | _ -> "string_of_typ no implementation yet."
```

sast.ml

```
(* signed: Yanlin Duan, Emily Meng, Shiyu Qiu *)

open Ast

type sexpr =
  | SNoexpr
  | SIntLit of int * typ
  | SBoolLit of bool * typ
  | SFloatLit of float * typ
  | SCharLit of char * typ
  | SArrayLit of sexpr list * int * typ
  | SStringLit of string * typ
  | SArrayAccess of sexpr * sexpr * typ
  | SStructCreate of ((string * sexpr) list)
  | SStructAccess of sexpr * string * typ
  | SId of string * typ
  | SBinop of sexpr * op * sexpr * typ
  | SUnop of uop * sexpr * typ
  | SCast of sexpr * typ
  | SCall of string * sexpr list * typ
  | SStructMethodCall of string * string * sexpr list * typ
```

```
type sstmt =
    SBlock of sstmt list
  | SDeclaration of borrowType * bind * sexpr
  | SExpr of sexpr * typ
  | SReturn of sexpr * typ
  | SBreak
  | SIf of sexpr * sstmt * sstmt
  | SFor of sexpr * sexpr * sexpr * sstmt
  | SWhile of sexpr * sstmt
  | SLoop of sstmt
  | SStructDef of string * bind list
  | SImplDef of string * sfunc_decl * typ

and sfunc_decl = {
    sfname : string;
    sformals : bind list;
    soutputType : typ;
    sbody : sstmt list;
  }

type program = sfunc_decl list

let get_type_from_sexpr = function
    SNoexpr -> DataT(VoidT)
  | SIntLit(_,t) -> t
  | SBoolLit(_,t) -> t
  | SFloatLit(_,t) -> t
  | SCharLit(_,t) -> t
  | SArrayLit(_,_,t) -> t
  | SStringLit(_,t) -> t
  | SArrayAccess(_,_,t) -> t
  | SStructCreate(_) -> StructT("unknown_struct")
  | SStructAccess(_,_,t) -> t
  | SId(_,t) -> t
  | SBinop(_,_,_,t) -> t
  | SUnop(_,_,t) -> t
  | SCall(_,_,t) -> t
  | SStructMethodCall(_,_,_,t) -> t
  | _ -> DataT(VoidT)

(* Pretty-printing functions *)

let rec string_of_sexpr = function
    SStringLit(s,_) -> s
  | SNoexpr -> ""
  | SIntLit(i,_) -> string_of_int i
  | SBoolLit(b,_) -> string_of_bool b
  | SFloatLit(f,_) -> string_of_float f
  | SCharLit(c,_) -> Char.escaped c
  | SCall(f, el,_) -> f ^ "(" ^ String.concat ", " (List.map string_of_sexpr
    el) ^ ")"
  | SId(a,_) -> a
  | SStructAccess(a,_,_)-> string_of_sexpr a
  | _ -> "string_of_sexpr not implemented yet"
```

```
let string_of_sstmt = function
    SDeclaration(_,(_,_),e) -> "s" ^ string_of_sexpr e
  | _ -> "string_of_sstmt not implemented yet"
```

pre_semant.ml

```
(* signed: Emily Meng *)

open Ast

module StringHash = Hashtbl.Make(struct
    type t = string (* type of keys *)
    let equal x y = x = y (* use structural comparison *)
    let hash = Hashtbl.hash (* generic hash function *)
end)

(* Preprocessing AST before passing to actual semant checker *)
(* ImplDef nodes converted into functions and removed from AST *)
(* StructMethodCall nodes converted into Call nodes *)

let check (functions) =

  let varMap = StringHash.create 20 in
  let implMap = StringHash.create 20 in
  let structMap = StringHash.create 20 in

  let check_function func =

    let rec expr_to_expr = function
        Noexpr                          -> Noexpr
      | IntLit(n)                       -> IntLit(n)
      | BoolLit(b)                      -> BoolLit(b)
      | FloatLit(f)                     -> FloatLit(f)
      | CharLit(c)                      -> CharLit(c)
      | StringLit(s)                    -> StringLit(s)
      | ArrayLit(exp)                   -> ArrayLit(exp)
      | Id(s)                           -> Id(s)
      | ArrayAccess(e1, e2)             -> ArrayAccess(e1, e2)
      | StructCreate(field_decl_list) -> StructCreate(field_decl_list)
      | StructAccess(e, s)              -> StructAccess(e, s)
      | StructMethodCall(s1, s2, el)  ->
        (* convert method call name to struct_name+method_name ex. f.sumXY to
           PointSumXY *)
        let struct_type =  StringHash.find varMap s1 in
          let new_fn_name = struct_type ^ s2 in
          (* pass in self as variable to actuals_list *)
          let new_fn_actuals = Id(s1) :: el in
            Call(new_fn_name, new_fn_actuals)
      | Unop(o, e) -> Unop(o, e)
      | Cast(e, t) -> Cast(e, t)
      | Binop(operand1, operator, operand2) -> Binop(operand1, operator,
        expr_to_expr operand2)
      | Call(fname, actuals) -> Call(fname, actuals)
```

```
  in

  let rec stmt_to_stmt = function
      Block(sl) ->
      let p_sl = List.map(fun x -> stmt_to_stmt x) sl in
      Block(p_sl)
    | Expr(e) ->
      let e' = expr_to_expr e in Expr(e')
    | Return(e)                 -> Return(e)
    | If(p, b1, b2)             -> If(p, b1, b2)
    | While(p, s)               -> While(p, s)
    | For(e1, e2, e3, st)       -> For(e1, e2, e3, st)
    | Declaration(b, (s, t), e) ->
      let t' = string_of_typ t in
      let e = expr_to_expr e in
        ignore(StringHash.add varMap s t');
        Declaration(b, (s, t), e)
    | StructDef(s, sl) ->
        StringHash.replace structMap s sl;
        StructDef(s,sl)
    | ImplDef(s, fdl) ->
      (*convert impl definition to function *)
      (*let func_impl = convert_impldef_to_fdecl s fdl in*)

      let func_impl =
        {
          (* impl Point { fn sumXY } -> fn PointsumXY *)
          fname = s ^ fdl.fname;
          (* arg list takes self as new arg in beginning *)
          formals = ("self", StructT(s)) :: fdl.formals;
          outputType = fdl.outputType;
          body = fdl.body;
        } in
        ignore(try StringHash.find structMap s
        with Not_found -> raise(Failure("Impl is associated with a
           non-defined struct!")));
        StringHash.add implMap (s ^ fdl.fname) func_impl;
        Expr(Noexpr)
    | _ -> Break

and convert_fdecl_to_fdecl fdecl =
{
  fname = fdecl.fname;
  formals = fdecl.formals;
  outputType = fdecl.outputType;
  body = List.map (fun x -> stmt_to_stmt x) fdecl.body;
}

in

convert_fdecl_to_fdecl func

in
```

```
    let func_list = List.map check_function functions in
    let list_arr = StringHash.fold (fun _ v l -> v :: l ) implMap [] in
    let all = List.append func_list list_arr in
    all
```

semant.ml

```
(* signed: Yanlin Duan, Emily Meng *)

open Ast
open Sast

module StringMap = Map.Make(String)

module StringHash = Hashtbl.Make(struct
    type t = string (* type of keys *)
    let equal x y = x = y (* use structural comparison *)
    let hash = Hashtbl.hash (* generic hash function *)
end)

(* Semantic checking of a program. Returns void if successful, throws an
   exception if something is wrong. Check each function *)

type symbol_table = {
  parent: symbol_table option;
  varMap: (borrowType*sexpr*typ*bool*bool*bool) StringHash.t;
}

type env = {
  scope : symbol_table;
  return_type : typ;
  in_for : bool;
  in_while : bool;
}


let check (functions) =

  let structMap = StringHash.create 20 in

  let check_function func =
    let rec findVariable scope name =
      try let a = StringHash.find scope.varMap name in (a,scope)
        with Not_found ->
          match scope.parent with
            Some(parent) -> findVariable parent name
          | _            -> raise (Failure ("undeclared identifier " ^ name))

  in

  let rec checkOwnership scope name =
    try let (_,_,_,own,_,_) = StringHash.find scope.varMap name in
      if own=false then raise (Failure ("use of moved value: " ^ name)) else
        ()
```

```
      with Not_found ->
        match scope.parent with
          Some(parent) -> checkOwnership parent name
        | _ -> raise (Failure ("undeclared identifier " ^ name))
in

let rec findDupVar scope name =
  try ignore(StringHash.find scope.varMap name);raise(Failure("undeclared
      identifier " ^ name))
    with Not_found ->
      match scope.parent with
        Some(parent) -> findDupVar parent name
      | _  -> ()
in

let rec toggleOwnership scope name =
  try let (b,e',t,_,x,y) = StringHash.find scope.varMap name in
     StringHash.replace scope.varMap name (b,e',t,false,x,y)
    with Not_found ->
      match scope.parent with
        Some(parent) -> toggleOwnership parent name
      | _ -> raise (Failure ("undeclared identifier " ^ name))
in

let move sexpr env =
  match sexpr with
     SId(s,_) -> checkOwnership env.scope s; toggleOwnership env.scope s;
   | _ -> ()
in

(* Raise an exception if the given list has a duplicate *)
let report_duplicate exceptf list =
  let rec helper = function
      n1 :: n2 :: _ when n1 = n2 -> raise (Failure(exceptf n1 ^ " duplicate
          variable"))
    | _ :: t -> helper t
    | [] -> ()
  in helper (List.sort compare list)
in

(* Raise an exception if a given binding is to a void type *)
let check_not_void exceptf = function
    (n,DataT(VoidT)) -> raise (Failure (exceptf n ^ "void type"))
  | _ -> ()
in

(* Raise an exception if the given rvalue type cannot be assigned to the
   given lvalue type *)
let check_assign lvaluet rvaluet err =
  if lvaluet = rvaluet then lvaluet
  else raise err
in

(* Raise an exception if the given struct declarations do not match the
```

```
    definitions *)
let check_fields formalt actualt =
  if formalt = actualt then formalt
  else raise (Failure ("illegal struct field found"))
in

let findMutability scope = function
    SId(n,_) -> let ((b,_,_,_,_,_),_) = findVariable scope n in (n,b)
  | _ -> raise(Failure("deal with the mutability check later!"))
in

let findName  = function
    SId(s,_) -> s
  | _ -> raise(Failure("findName not implemented yet!"))
in

let rec toggleBorrow scope name hasImmutBorrow hasMutBorrow =
  try let (b,e',t,own,_,_) = StringHash.find scope.varMap name in
    StringHash.replace scope.varMap name
       (b,e',t,own,hasImmutBorrow,hasMutBorrow)
    with Not_found ->
      match scope.parent with
          Some(parent) -> toggleBorrow parent name hasImmutBorrow
              hasMutBorrow
        | _ -> raise (Failure ("undeclared identifier " ^ name))
in

(* Raise an exception if binary operator does not support operations
   between operand types *)
let check_binop_type env operand1 operator operand2 =
  match operator with
      Add | Sub | Mult  ->
      (match (get_type_from_sexpr operand1, get_type_from_sexpr operand2)
          with
          (DataT(IntT), DataT(IntT))      -> SBinop(operand1, operator,
              operand2, DataT(IntT))
        | (DataT(FloatT), DataT(FloatT)) -> SBinop(operand1, operator,
            operand2, DataT(FloatT))
        | (DataT(IntT), DataT(FloatT))    ->
            (match operand1 with
                SIntLit(n, _) -> SBinop(SFloatLit(float_of_int n,
                    DataT(FloatT)), operator, operand2, DataT(FloatT))
              | _             -> SBinop(operand1, operator, operand2,
                  DataT(FloatT))
            )
        | (DataT(FloatT), DataT(IntT))    ->
            (match operand2 with
                SIntLit(n, _) -> SBinop(operand1, operator,
                    SFloatLit(float_of_int n, DataT(FloatT)), DataT(FloatT))
              | _             -> SBinop(operand1, operator, operand2,
                  DataT(FloatT))
            )
        | _ -> raise (Failure(string_of_op operator ^ " does not support
            operations between " ^ string_of_sexpr operand1 ^ " and " ^
```

```
      string_of_sexpr operand2)) )
| Div | Mod ->
  (match (get_type_from_sexpr operand1, get_type_from_sexpr operand2)
     with
      (DataT(IntT), DataT(IntT)) -> (match operand2 with
          SIntLit(0, _) -> raise (Failure("division by zero!"))
        | _                          -> SBinop(operand1,
          operator, operand2, DataT(IntT)))
    | (DataT(FloatT), DataT(FloatT))
    | (DataT(IntT), DataT(FloatT)) | (DataT(FloatT), DataT(IntT)) ->
      (match operand2 with
        SIntLit(0, _) | SFloatLit(0.0, _) -> raise (Failure("division
           by zero!"))
        | _                          -> SBinop(operand1,
          operator, operand2, DataT(FloatT)))
    | _ -> raise (Failure(string_of_op operator ^ " does not support
      operations between " ^
                string_of_sexpr operand1 ^ "and" ^string_of_sexpr
                   operand2)) )
| Equal | Neq | Less | Leq | Greater | Geq ->
  (match (get_type_from_sexpr operand1, get_type_from_sexpr operand2)
     with
      (DataT(BoolT), DataT(BoolT))  -> SBinop(operand1, operator,
         operand2, DataT(BoolT))
    | (DataT(IntT),DataT(IntT))     -> SBinop(operand1, operator,
      operand2, DataT(BoolT))
    | (DataT(FloatT),DataT(FloatT)) -> SBinop(operand1, operator,
      operand2, DataT(BoolT))
    | _ -> raise (Failure(string_of_op operator ^ " does not support
      operations between these operators " ^
                string_of_sexpr operand1 ^ " and " ^ string_of_sexpr
                   operand2)))
| And | Or ->
  (match (get_type_from_sexpr operand1, get_type_from_sexpr operand2)
     with
      (DataT(BoolT), DataT(BoolT)) -> SBinop(operand1, operator,
         operand2, DataT(BoolT))
    | _ -> raise (Failure(string_of_op operator ^ " does not support
      operations between these operators " ^
                string_of_sexpr operand1 ^ " and " ^ string_of_sexpr
                   operand2)) )
| AS -> raise(Failure("AS not supported"))
| Assign -> (match operand1 with
  | SNoexpr | SIntLit(_,_) | SBoolLit(_,_) | SFloatLit(_,_) |
    SCharLit(_,_) | SArrayLit(_,_,_) | SStringLit(_,_) | SCast(_,_) |
    SCall(_,_,_) | SBinop(_,_,_,_) |SUnop(Neg,_,_) | SUnop(Not,_,_)
    |SUnop(Borrow(_),_,_) -> raise (Failure(string_of_sexpr operand1
    ^ "cannot be used as lvalue!"))
  | _ -> let t = get_type_from_sexpr operand1 and rt =
    get_type_from_sexpr operand2 in
        ignore(check_assign t rt (Failure ("illegal assignment " ^
           string_of_typ t ^ " = " ^ string_of_typ rt ^ " in " ^
        string_of_sexpr
           (SBinop(operand1,operator,operand2,get_type_from_sexpr
```

```
                operand2)))));
            let (s,b) = (match operand1 with
                  SId(s,_) -> let ((b,_,_,_,_,_),_) = findVariable
                      env.scope s in (s,b)
                | SUnop(Deref,se,_) -> findMutability env.scope se
                | SStructAccess(_, _, _) -> ("a", Mut)
                | _ -> raise(Failure("not supported"))) in

            (match b with
            | Immut -> raise (Failure ("trying to modify an immutable
                value" ^ s))
            | _ -> ());

            (match operand2 with
                SId(s,_) -> checkOwnership env.scope s
            | _ -> ());

            SBinop(operand1,operator,operand2,get_type_from_sexpr
                operand2);
          )
in

(**** Checking functions ****)

(* Check overriding standard library functions *)
if List.mem "println" (List.map (fun fd -> fd.fname) functions)
  then raise (Failure ("function println cannot be overwritten!"))
else ();

(* Check duplicate function declarations *)
report_duplicate (fun n -> "duplicate function " ^ n) (List.map (fun fd ->
    fd.fname) functions);

(* Standard library function declarations *)
let built_in_decls =  StringMap.singleton "println"
  {
    outputType = DataT(VoidT);
    fname = "println";
    formals = [("s",StringT)];
    body = []
  }
in

(* Add standard library functions to function_decls map *)
let function_decls = List.fold_left (fun m fd -> StringMap.add fd.fname fd
    m) built_in_decls functions
in

(* Check function exists in function_decls map *)
let function_decl s = try StringMap.find s function_decls
  with Not_found -> raise (Failure ("unrecognized function " ^ s))
in

(* Ensure "main" is defined *)
```

```
let _ = function_decl "main"
in

(* Verify an expression or throw an exception *)
let rec expr_to_sexpr env = function
    Noexpr -> SNoexpr
  | IntLit(n) -> SIntLit(n,DataT(IntT))
  | BoolLit(b) -> SBoolLit(b,DataT(BoolT))
  | FloatLit(f) -> SFloatLit(f,DataT(FloatT))
  | CharLit(c) -> SCharLit(c,DataT(CharT))
  | StringLit(s) -> SStringLit(s,StringT)
  | ArrayLit(exp) ->
    let rec iter t sel = function
        []      -> sel, t
      | e :: el ->
          let se = expr_to_sexpr env e in
            let se_t = get_type_from_sexpr se in
              if t = se_t
              then iter t (se :: sel) el
              else raise (Failure ("multiple types in array " ^
                 string_of_expr (ArrayLit(exp))))
    in
    let se = expr_to_sexpr env (List.hd exp) in
      let el = List.tl exp in
        let se_t = get_type_from_sexpr se in
          let sel, t = iter se_t ([se]) el in
            SArrayLit(List.rev sel, List.length sel, ArrayT(t,List.length
               sel))


  | Id(s) -> checkOwnership env.scope s; let ((_,_,t,_,_,_),_) =
     findVariable env.scope s in SId(s,t)

  | ArrayAccess(e1, e2) -> let e1' = expr_to_sexpr env e1 in
    (match get_type_from_sexpr e1' with
        ArrayT(t,_) -> SArrayAccess(e1', expr_to_sexpr env e2, t)
      | _ -> raise(Failure("semant line 268, array access error")))

  | StructCreate(field_decl_list) ->
    (* check lists of ids for duplicates *)
    let name_list = List.map fst field_decl_list in
      report_duplicate (fun n -> "field " ^ n) name_list;

    let arg_list = List.map snd field_decl_list in
      let s_arg_list = List.map (expr_to_sexpr env) (arg_list) in

    let s_field_decl_list = List.combine name_list s_arg_list in
    SStructCreate(s_field_decl_list)

  | StructAccess(e,s) ->
    (* check for var existance *)
    let e' = expr_to_sexpr env e in
      let et' = get_type_from_sexpr e' in
        let ((_, _, t, _,_,_),_) = findVariable env.scope (findName e') in
```

```
        (* check arg struct type with returned type t *)
        let _ = check_fields t et' in

        (*check struct type has field variable name s *)
        let fld_list = try StringHash.find structMap (string_of_typ t)
          with Not_found -> raise(Failure("undeclared struct " ^
              string_of_typ t)) in
          let name_fld_list = List.map fst fld_list in
              let b = List.mem s name_fld_list in
                if b
                then let fld_type = List.assoc s fld_list in
                  SStructAccess(e',s,fld_type);
                else raise(Failure("no field " ^ s ^ " in struct type " ^
                    string_of_typ t))

| Unop(o, e) -> let e' = expr_to_sexpr env e in
  let oldt = get_type_from_sexpr e' in
    let newt = (match o with
      | Neg ->
        (match get_type_from_sexpr e' with
            DataT(IntT) -> oldt
          | DataT(FloatT) -> oldt
          | _ -> raise(Failure("don't support this operator in " ^
              string_of_sexpr e')))
      | Not ->
        (match get_type_from_sexpr e' with
            DataT(BoolT) -> oldt
          | _ -> raise(Failure("don't support not in " ^ string_of_sexpr
              e')))
      | Deref -> (match oldt with
          RefT(bt,t) ->  let name = findName e' in
          let ((_,_,_,_,hasImmutBorrow,hasMutBorrow),scope) =
              findVariable env.scope name in
            (match bt with
                Immut -> if hasMutBorrow = true then
                    raise(Failure("mutable and immutable borrow at the
                    same time!")) else
                     toggleBorrow scope name true false
              | Mut -> if hasImmutBorrow = true then
                  raise(Failure("mutable and immutable borrow at the same
                  time!")) else ();
                  toggleBorrow scope name false true); t
        | _ -> raise(Failure("Dereferencing a non-borrow at " ^
            string_of_sexpr e')))
      | Borrow(bt) ->
        if snd (findMutability env.scope e') = Immut && bt = Mut then
            raise(Failure("cannot mutably borrow a imuutable value!" ^
            string_of_sexpr e')) else ();

          let name = findName e' in
          let ((_,_,_,_,hasImmutBorrow,hasMutBorrow),scope) =
              findVariable env.scope name in
            (match bt with
                Immut -> if hasMutBorrow = true then
```

```
                    raise(Failure("mutable and immutable borrow at the
                    same time!")) else
                     toggleBorrow scope name true false
              | Mut -> if hasMutBorrow = true then raise(Failure("at most
                 one mutable borrow!")) else ();
                 if hasImmutBorrow = true then raise(Failure("mutable and
                    immutable borrow at the same time!")) else ();
                 toggleBorrow scope name false true);
            RefT(bt,oldt)) in
  SUnop(o, e', newt)
| Cast(e, t) -> SCast(expr_to_sexpr env e, t)
| Binop(operand1, operator, operand2) ->
    let so1 = expr_to_sexpr env operand1 and so2 = expr_to_sexpr env
       operand2 in
     check_binop_type env so1 operator so2
| Call(fname, actuals) as call ->
  (match fname with
    | "println" ->
      if List.length actuals != 1
      then raise (Failure ("expecting 1 arguments in println!"))
      else
        let actuals' = expr_to_sexpr env (List.hd actuals) in let
           actualsType = get_type_from_sexpr actuals' in
          (match actualsType with
            DataT(_) | StringT ->
              let fd = function_decl "println" in
              SCall(fname,List.map (fun x -> expr_to_sexpr env x)
                 actuals, fd.outputType)
          | _ -> raise (Failure ("illegal actual argument found " ^
              string_of_typ actualsType ^ " in println")))
    | _ -> let fd = function_decl fname in
      if List.length actuals <> List.length fd.formals
      then raise (Failure ("expecting " ^ string_of_int (List.length
         fd.formals) ^ " arguments in " ^ string_of_expr call))
      else
        List.iter2 (fun (_, ft) e -> let e' = expr_to_sexpr env e in
          let et = get_type_from_sexpr e' in
            ignore (check_assign ft et (Failure ("illegal actual argument
               found " ^ string_of_typ et ^ " expected " ^ string_of_typ
               ft ^ " in " ^ fname)))) fd.formals actuals;
        let sactuals = List.map (fun x -> expr_to_sexpr env x)  actuals in
        List.iter (fun x -> move x env) sactuals;
        SCall(fname,sactuals,fd.outputType))
| _ -> SNoexpr

in

let check_bool_expr env e =
  let e' = expr_to_sexpr env e in
    let et = get_type_from_sexpr e' in
      let boolt = get_type_from_sexpr(SBoolLit(true, DataT(BoolT))) in
        ignore (check_assign boolt et (Failure ("expected Boolean
           expression in " ^ string_of_sexpr e'))) in
```

```
(* Verify a statement or throw an exception *)
let rec stmt_to_sstmt env = function
    Block(sl) ->
      let new_env =
      {
        env with scope =
          {
            parent = Some(env.scope);
            varMap = StringHash.create 20
          }
      } in SBlock(List.map (fun x -> stmt_to_sstmt new_env x) sl)
  | Expr(e) -> let e' = expr_to_sexpr env e in let et =
     get_type_from_sexpr e' in SExpr(e',et)
  | Return(e) -> let e' = expr_to_sexpr env e in let t =
     get_type_from_sexpr e' in
    if t = func.outputType
    then ()
    else
      raise (Failure ("return gives " ^ string_of_typ t ^ " expected " ^
         string_of_typ func.outputType ^ " in " ^ string_of_expr e));
      SReturn(e',t)
  | If(p, b1, b2) -> check_bool_expr env p;
    SIf(expr_to_sexpr env p,stmt_to_sstmt env b1,stmt_to_sstmt env b2)
  | While(p, s) -> check_bool_expr env p;
    let new_env =
    {
      env with scope =
        {
          parent = Some(env.scope);
          varMap = StringHash.create 20
        };
      in_while = true;
    } in SWhile(expr_to_sexpr new_env p, stmt_to_sstmt new_env s)
  | For(e1, e2, e3, st) -> SFor(expr_to_sexpr env e1, expr_to_sexpr env
     e2, expr_to_sexpr env e3, stmt_to_sstmt env st)
  | Declaration(b, (s, oldt), e) ->
    let e' = expr_to_sexpr env e in let rt = get_type_from_sexpr e' in
        (* If assigmment is valid, put that / replace that in varMap, and
           return the SDecl *)
        let t =
          (match oldt with
            ArrayT(_,_) -> ignore(check_assign oldt rt (Failure ("illegal
               assignment " ^ string_of_typ oldt ^ " = " ^ string_of_typ
               rt ^ " in " ^ s))); oldt
          | ArrayTD(lt,ns) -> checkOwnership env.scope ns; let
             ((_,var,t,_,_,_),_) = findVariable env.scope ns in
                      if t <> DataT(IntT) then raise (Failure (ns ^ "
                         should be an int!")) else ();
                      let var = (match var with
                      SIntLit(var,_) -> var
                      | _ -> raise (Failure (ns ^ " should be an
                         int!")))
                       in let lt = ArrayT(lt,var) in
                      ignore(check_assign lt rt (Failure ("illegal
```

```
                    assignment " ^ string_of_typ oldt ^ " = " ^
                    string_of_typ rt ^ " in " ^ s ))); rt
      | StructT(_) ->
          (* check struct definition exists *)
          let sfields_list = try StringHash.find structMap
              (string_of_typ oldt)
            with Not_found -> raise(Failure("undeclared struct " ^
                string_of_typ oldt)) in
          (* check duplicate variable name *)
          ignore(findDupVar env.scope s);

          (* check lists of ids *)
          let def_name_list = List.map fst sfields_list in
          (*let def_type_list = List.map snd sfields_list in*)
            (match e' with
                SStructCreate(a) ->
                  let name_list = List.map fst a in
                  let _ = try List.map2 check_fields def_name_list
                      name_list
                  with Invalid_argument(_) -> raise (Failure("number
                      of struct fields do not match!")) in ()
              | _ -> () );

          (* check lists of args *)
          let def_type_list = List.map snd sfields_list in
            (match e' with
                SStructCreate(a) ->
                  let arg_list = List.map snd a in
                    let type_list = List.map get_type_from_sexpr
                        arg_list in
                    let _ = try List.map2 check_fields def_type_list
                        type_list
                    with Invalid_argument(_) -> raise
                        (Failure("number of struct field types do not
                        match!")) in ()
              | _ -> () );
            oldt
      | _ ->

        (match oldt with
          RefT(bt,_)-> if bt<>b then raise(Failure("borrow type
              unmatched at " ^ s )) else ()
        | _ -> ());
        ignore(check_assign oldt rt (Failure ("illegal assignment " ^
            string_of_typ oldt ^ " = " ^ string_of_typ rt ^ " in " ^
            s))); oldt) in
      move e' env;
      StringHash.add env.scope.varMap s (b, e', t,true, false,false);
      SDeclaration(b,(s, t),e');
| StructDef(s, sl) ->
  (* check for already defined struct *)
  let _ = try ignore(StringHash.find structMap s);
      raise(Failure("already defined struct name " ^ s))
      with Not_found ->
```

```
            (* check for duplicate fields *)
            StringHash.add structMap s sl;
          in
        SStructDef(s,sl)
    | _ -> SBreak

  and convert_fdecl_to_sfdecl env fdecl =
  {
      sfname = fdecl.fname;
      sformals = fdecl.formals;
      soutputType = fdecl.outputType;
      sbody = List.map (fun x -> stmt_to_sstmt env x) fdecl.body;
  } in

  List.iter (check_not_void (fun n -> "illegal void formal " ^ n ^
    " in " ^ func.fname)) func.formals;

  report_duplicate (fun n -> "duplicate formal " ^ n ^
    " in " ^ func.fname) (List.map fst func.formals);

  let varMap = StringHash.create 20 in
     List.iter (fun (s,t) -> let b = match t with
                   RefT(bt,_) -> bt
                 | _          -> Immut in StringHash.add varMap s (b, SId(s,
                   t), t, true, false,false)) func.formals;
  let env =
      {
        scope = {parent = None; varMap = varMap};
        return_type = func.outputType;
        in_for = false;
        in_while = false;
      } in

  convert_fdecl_to_sfdecl env func; in

  List.map check_function functions
```

codegen.ml

```
(* signed: Yanlin Duan, Zhuo Kong, Emily Meng, Shiyu Qiu *)

(* Code generation: translate takes a semantically checked AST and
produces LLVM IR

LLVM tutorial: Make sure to read the OCaml version of the tutorial

http://llvm.org/docs/tutorial/index.html

Detailed documentation on the OCaml LLVM library:

http://llvm.moe/
http://llvm.moe/ocaml/

*)
```

```
module L = Llvm
module A = Ast
module S = Sast
module E = Semant
open Ast
open Llvm
open Sast

module StringMap = Map.Make(String)

module StringHash = Hashtbl.Make(struct
    type t = string (* type of keys *)
    let equal x y = x = y (* use structural comparison *)
    let hash = Hashtbl.hash (* generic hash function *)
end)

type symbol_table = {
  parent: symbol_table option;
  varMap: (llvalue*typ) StringHash.t;
}

type env = {
  scope : symbol_table;
  return_type : typ;
  in_for : bool;
  in_while : bool;

}

let translate (functions) =
  let context = L.global_context () in
  let the_module = L.create_module context "rusty"
  and i32_t  = L.i32_type context
  and i8_t   = L.i8_type context
  and i1_t   = L.i1_type context
  and float_t = L.double_type context
  and string_t = (L.pointer_type (L.i8_type context))
  and void_t = L.void_type context in

let structMap = StringHash.create 20 in
  let ltype_of_primitive = (function
      A.IntT -> i32_t
    | A.BoolT -> i1_t
    | A.FloatT -> float_t
    | A.CharT -> i8_t
    | A.VoidT -> void_t
  ) in

let rec ltype_of_typ = (function
      A.DataT(t) -> ltype_of_primitive t
    | A.StringT -> string_t
    | A.ArrayT(t,_) -> L.pointer_type (ltype_of_typ t)
    | A.RefT(_,t) ->  L.pointer_type (ltype_of_typ t)
```

```
    | A.StructT(t) -> (try let t = snd (StringHash.find structMap t) in
       L.pointer_type t with Not_found -> raise(Failure(t)))
    | _ -> void_t
  ) in

let cast lhs rhs lhsType rhsType =

  (match (lhsType, rhsType) with
    (DataT(IntT), DataT(IntT)) -> (lhs, rhs), DataT(IntT)
  | (DataT(IntT), DataT(CharT))-> (lhs, rhs), DataT(CharT)
  | (DataT(IntT), DataT(FloatT))-> (lhs, rhs), DataT(FloatT)
  | (DataT(CharT), DataT(IntT))-> (lhs, rhs), DataT(CharT)
  | (DataT(CharT), DataT(CharT))-> (lhs, rhs), DataT(CharT)
  | (DataT(BoolT), DataT(BoolT))-> (lhs, rhs), DataT(BoolT)
  | (DataT(FloatT), DataT(IntT)) -> (lhs, rhs), DataT(FloatT)
  | (DataT(FloatT), DataT(FloatT)) -> (lhs, rhs), DataT(FloatT)
  |    _ -> raise(Failure("cannot support other types"))) in

  (* Declare printf(), which the print built-in function will call *)
  let printf_t = L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
  let printf_func = L.declare_function "printf" printf_t the_module in

   let rec prestmt = function
       S.SBlock(sl) -> List.iter prestmt sl
     | S.SStructDef(struct_name, l) ->
           let struct_fields = List.mapi (fun i (field, t) ->
               let offset = i * 4 in
                   (field, (t, offset))) l in
           let type_list = List.map snd l in
           let l_type_list = List.map ltype_of_typ type_list in
           let ltype = struct_type context (Array.of_list l_type_list) in
         StringHash.replace structMap struct_name (struct_fields, ltype)
     | _ -> () in

  (* Define each function (arguments and return type) so we can call it *)
  let function_decls =
    let function_decl m fdecl =
    let name = fdecl.S.sfname
      and formal_types =
        Array.of_list (List.map (fun (_,t) -> ltype_of_typ t)
           fdecl.S.sformals) in
        prestmt (S.SBlock(fdecl.S.sbody));
        let ftype = L.function_type (ltype_of_typ fdecl.S.soutputType)
           formal_types in
          StringMap.add name (L.define_function name ftype the_module, fdecl)
            m in
          List.fold_left function_decl StringMap.empty functions in

  (* Fill in the body of the given function *)
  let build_function_body fdecl =
    let varMap = StringHash.create 20 in

    let (the_function, _) = StringMap.find fdecl.S.sfname function_decls in
    let builder = L.builder_at_end context (L.entry_block the_function) in
```

```
    let add_variable m (n, t) v =
    let var = (match t with
        A.StructT(s) -> let (_,lt) = StringHash.find structMap s in
          let ltyp = (pointer_type lt) in
          let var = L.build_alloca ltyp n builder in ignore (L.build_store
              v var builder); var
      | A.StringT -> v
      | _ -> let ltyp = ltype_of_typ t in
          let var = L.build_alloca ltyp n builder in ignore (L.build_store
              v var builder); var) in
    StringHash.replace m n (var,t)
in

let _ = List.iter2 (fun a b-> add_variable varMap a b)
   fdecl.S.sformals(Array.to_list (L.params the_function)) in

let rec lookup scope name =
  try
    StringHash.find scope.varMap name;
  with Not_found ->
      match scope.parent with
      Some(parent) -> lookup parent name
    | _ -> raise (Failure ("undeclared identifier " ^ name))
in

let env = {
  scope = {parent = None; varMap = varMap};
  return_type = fdecl.S.soutputType;
  in_for = false;
  in_while = false;
} in

let rec search_index l field =
  match l with
    [] -> 0
  | hd :: tl -> if field = hd then 0 else 1 + search_index tl field in

let int_format_str = L.build_global_stringptr "%d\n" "imt" builder in
let float_format_str = L.build_global_stringptr "%0.2f\n" "fmt" builder in
let char_format_str = L.build_global_stringptr "%e\n" "cmt" builder in
let string_format_str = L.build_global_stringptr "%s\n" "sfmt" builder in

  (* Construct code for an expression; return its value *)
  let rec expr builder env = function
      S.SIntLit(i,_) -> L.const_int i32_t i
    | S.SBoolLit(b,_) -> L.const_int i1_t (if b then 1 else 0)
    | S.SFloatLit(f,_) -> L.const_float float_t f
    | S.SCharLit(c,_) -> L.const_int i8_t (Char.code c)
    | S.SStringLit(s,_) -> L.build_global_stringptr (s) "tmp" builder
    | S.SCall ("println", [e],_) ->

      (match (S.get_type_from_sexpr e) with
```

```
  StringT ->
        (match e with
                  S.SStringLit(s,_) -> let strptr =
                      L.build_global_stringptr (s^"\n") "printstr"
                      builder in
                  L.build_call printf_func [| strptr |] "printf" builder
         |        S.SId(var,_) -> let (strptr, _) = (StringHash.find
             env.scope.varMap var) in
                  L.build_call printf_func [|string_format_str; strptr
                      |] "printf" builder
         |        S.SArrayAccess(_,_,_) -> let strptr = expr builder
             env e in
                  L.build_call printf_func [|string_format_str; strptr
                      |] "printf" builder
         | _ -> raise(Failure("not implemented" ))
        )
    | DataT(IntT) -> L.build_call printf_func [| int_format_str ; (expr
       builder env e)|] "printf" builder
    | DataT(FloatT) -> L.build_call printf_func [| float_format_str ;
       (expr builder env e)|] "printf" builder
    | DataT(CharT) -> L.build_call printf_func [| char_format_str ;
       L.build_call printf_func [| char_format_str ; (expr builder env
       e)|]
      "printf" builder|] "printf" builder
    | _ -> raise(Failure("println does not support types other than int
       and string yet!"))  )

  | S.SCall (fname, args, _) ->
     let (fdef, fdecl) = StringMap.find fname function_decls in

     let actuals = List.rev (List.map (expr builder env) (List.rev args))
        in

     let result = match fdecl.S.soutputType with DataT(VoidT) -> ""
                                                 | _ -> fname ^ "_result" in
          let x = L.build_call fdef (Array.of_list actuals) result
             builder in
          x

   | S.SBinop(e1,op,e2,_) ->
      let type1 =get_type_from_sexpr e1 in
      let type2 = get_type_from_sexpr e2 in
      let e1' = expr builder env e1 and e2' = expr builder env e2 in

      let float_ops op e1 _ =
      (match op with
          A.Add     -> L.build_fadd e1' e2' "tmp" builder
        | A.Sub     -> L.build_fsub e1' e2' "tmp" builder
        | A.Mult    -> L.build_fmul e1' e2' "tmp" builder
        | A.Div     -> L.build_fdiv e1' e2' "tmp" builder
        | A.Mod     -> L.build_frem e1' e2' "tmp" builder
        | A.Equal   -> L.build_fcmp L.Fcmp.Oeq e1' e2' "tmp" builder
        | A.Neq     -> L.build_fcmp L.Fcmp.One e1' e2' "tmp" builder
        | A.Less    -> L.build_fcmp L.Fcmp.Ult e1' e2' "tmp" builder
```

```
      | A.Leq     -> L.build_fcmp L.Fcmp.Ole e1' e2' "tmp" builder
      | A.Greater -> L.build_fcmp L.Fcmp.Ogt e1' e2' "tmp" builder
      | A.Geq     -> L.build_fcmp L.Fcmp.Oge e1' e2' "tmp" builder
      | A.Assign -> (match e1 with
            S.SId(s,_) -> ignore (L.build_store e2' (fst (lookup
               env.scope s)) builder); e2'
          | S.SUnop(Deref,e,_) -> let e1'' = expr builder env e in
            ignore (L.build_store e2' e1'' builder); e2'
          | SStructAccess(e, _, _) -> let e1'' = expr builder env e in
            let e1'' = build_pointercast e1'' (pointer_type float_t)
               "tmp" builder
            in ignore (L.build_store e2' e1'' builder); e2'
          | _ -> raise(Failure("Float doesn't support this
            operator!")))
      |_ -> raise(Failure("Float doesn't support this operator!"))
  ) in
  let int_ops op e1 _ =
      (match op with
        A.Add     -> L.build_add e1' e2' "addtmp" builder
      | A.Sub     -> L.build_sub e1' e2' "tmp" builder
      | A.Mult    -> L.build_mul e1' e2' "tmp" builder
      | A.Div     -> L.build_sdiv e1' e2' "tmp" builder
      | A.Mod     -> L.build_srem e1' e2' "tmp" builder
      | A.And     -> L.build_and e1' e2' "tmp" builder
      | A.Or      -> L.build_or e1' e2' "tmp" builder
      | A.Equal   -> L.build_icmp L.Icmp.Eq e1' e2' "tmp" builder
      | A.Neq     -> L.build_icmp L.Icmp.Ne e1' e2' "tmp" builder
      | A.Less    -> L.build_icmp L.Icmp.Slt e1' e2' "tmp" builder
      | A.Leq     -> L.build_icmp L.Icmp.Sle e1' e2' "tmp" builder
      | A.Greater -> L.build_icmp L.Icmp.Sgt e1' e2' "tmp" builder
      | A.Geq     -> L.build_icmp L.Icmp.Sge e1' e2' "tmp" builder
      | A.Assign -> (match e1 with
            S.SId(s,_) -> ignore (L.build_store e2' (fst (lookup
               env.scope s)) builder); e2'
          | S.SUnop(Deref,e,_) -> let e1'' = expr builder env e in
            ignore (L.build_store e2' e1'' builder); e2'
          | _ -> raise(Failure("int doesn't support this operator!")))

      | _ -> raise(Failure("int doesn't support this operator!")))
   in

  let (e1,e2),d = cast e1 e2 type1 type2 in
  let type_handler d =(match d with
      A.DataT(FloatT) ->float_ops op e1 e2
    | A.DataT(IntT)
    | A.DataT(BoolT)
    | A.DataT(CharT)  ->int_ops op e1 e2
    | _ -> raise(Failure("no type matched!"))
  ) in type_handler d

| S.SUnop(op, e, _) -> let e' = expr builder env e in
    (match op with
      A.Neg     -> L.build_neg e' "tmp" builder
    | A.Not     -> L.build_not e' "tmp" builder
```

```
        | A.Borrow(_) -> (match e with
                        SId(s,_) -> (fst (lookup env.scope s))
                        | _ -> raise(Failure("no type matched!")))
        | A.Deref  -> L.build_load e' "tmp" builder)
  | S.SId(s,_) -> L.build_load (fst (lookup env.scope s)) s builder
  | S.SArrayLit(l, length, tp) -> array_create l length tp env builder
  | S.SArrayAccess(name, index, _) -> array_access name index env builder
  | S.SStructCreate(fields) -> struct_create fields env builder
  | S.SStructAccess(var, field, _) -> struct_access var field env builder
  | _ -> L.const_int i32_t 0

and array_create l length tp env builder =
match tp with
  ArrayT(tp,_) -> let t = ltype_of_typ tp in
  let length_used = (const_int i32_t (length  + 1)) in
    let a = build_array_malloc t length_used "tmp" builder in
    let a = build_pointercast a (pointer_type t) "tmp" builder in
    let llvalues = List.map (expr builder env) l in
      List.iteri (fun i llval ->
            let ptr = build_gep a [| (const_int i32_t (i+1)) |] "tmp"
               builder in
            ignore(build_store llval ptr builder);) llvalues;
      a
| _ -> raise(Failure("array error"))

and array_access name idx env builder =
  let idx = expr builder env idx in
    let idx = build_add idx (const_int i32_t 1) "tmp" builder
    in
      let arr = expr builder env name in
      let _val = build_gep arr [| idx |] "tmp" builder in
      build_load _val "tmp" builder

and struct_create fields env builder =

    let arg_list = List.map snd fields in
    let l_type_list = List.map ltype_of_typ (List.map
       S.get_type_from_sexpr arg_list) in
      let ltype = struct_type context (Array.of_list l_type_list) in
        let v = build_malloc ltype "tmp" builder in
      List.iteri(fun i arg->
      let field_ptr = build_struct_gep v i "tmp" builder in
          ignore(build_store ((expr builder env) arg) field_ptr builder))
             arg_list; v

and struct_access var field env builder =
  let string_var = S.string_of_sexpr var in
    let (struct_ptr,t) = lookup env.scope string_var in
    let (struct_l,_) = try StringHash.find structMap (A.string_of_typ t)
      with Not_found -> raise(Failure("codegen undeclared struct " ^
         (A.string_of_typ t))) in
    let fields_list = List.map fst struct_l in
    let index = search_index fields_list field in
    let struct_ptr = L.build_load struct_ptr "tmp" builder in
```

```
      let field_ptr = build_struct_gep struct_ptr index "tmp" builder in

      L.build_load field_ptr "tmp" builder
in
(* Invoke "f builder" if the current block doesn't already
   have a terminal (e.g., a branch). *)
let add_terminal builder f =
  match L.block_terminator (L.insertion_block builder) with
    Some _ -> ()
  | None -> ignore (f builder) in

(* Build the code for the given statement; return the builder for
   the statement's successor *)
let rec stmt builder env = function
    S.SBlock(sl) -> let new_env = {env with scope =
        {parent=Some(env.scope); varMap = StringHash.create 20}} in
    let builder = List.fold_left (fun a b -> stmt a new_env b) builder sl
        in
    builder
  | S.SExpr(e,_) -> ignore (expr builder env e); builder
  | S.SReturn(e,_) -> ignore (match fdecl.S.soutputType with
        A.DataT(A.VoidT) -> L.build_ret_void builder
      | _ -> L.build_ret (expr builder env e) builder); builder
  | S.SIf (pred_expr, then_stmt, else_stmt) ->
    let bool_val = expr builder env pred_expr in
    let merge_bb = L.append_block context "merge" the_function in
    let then_bb = L.append_block context "then" the_function in
    let new_env = {env with scope = {parent=Some(env.scope); varMap =
        StringHash.create 20}} in
      add_terminal (stmt (L.builder_at_end context then_bb) new_env
          then_stmt)
      (L.build_br merge_bb);

    let else_bb = L.append_block context "else" the_function in
    let new_env = {env with scope = {parent=Some(env.scope); varMap =
        StringHash.create 20}} in
      add_terminal (stmt (L.builder_at_end context else_bb) new_env
          else_stmt)
      (L.build_br merge_bb);

    ignore (L.build_cond_br bool_val then_bb else_bb builder);
    L.builder_at_end context merge_bb
  | S.SWhile (pred_expr, body_stmt) ->
    let pred_bb = L.append_block context "while" the_function in
    ignore (L.build_br pred_bb builder);

    let body_bb = L.append_block context "while_body" the_function in
    let new_env = {env with scope = {parent=Some(env.scope); varMap =
        StringHash.create 20} ; in_while = true;} in
      add_terminal (stmt (L.builder_at_end context body_bb) new_env
          body_stmt)
      (L.build_br pred_bb);

    let pred_builder =  L.builder_at_end context pred_bb in
```

```
        let bool_val = expr pred_builder env pred_expr in
        let merge_bb = L.append_block context "merge" the_function in

        ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder);
        L.builder_at_end context merge_bb
      | S.SFor (e1, e2, e3, body) ->
        stmt builder env
        ( S.SBlock [S.SExpr (e1, S.get_type_from_sexpr e1) ; S.SWhile (e2,
           S.SBlock [body ; S.SExpr (e3, S.get_type_from_sexpr e3)]) ] )
      | S.SLoop (body) -> let new_env = {env with scope =
        {parent=Some(env.scope); varMap = StringHash.create 20}; in_while=
        true;} in
        stmt builder new_env (S.SBlock [S.SWhile
           (S.SBoolLit(true,A.DataT(A.BoolT)), S.SBlock [body;])])
      | S.SStructDef(struct_name, l) ->
          let struct_fields = List.mapi (fun i (field, t) ->
              let offset = i * 4 in
                  (field, (t, offset))) l in

          let type_list = List.map snd l in
          let l_type_list = List.map ltype_of_typ type_list in
          let ltype = struct_type context (Array.of_list l_type_list) in

        StringHash.replace structMap struct_name (struct_fields, ltype);
        builder
      | S.SDeclaration (_,(s,t),e) -> let e' = expr builder env e in
        add_variable env.scope.varMap (s,t) e'; builder
      | _ -> builder

    in

    (* Build the code for each statement in the function *)
    let builder = stmt builder env (S.SBlock fdecl.S.sbody) in

    match fdecl.S.soutputType with
        A.DataT(A.VoidT) -> add_terminal builder (L.build_ret_void)
      | A.DataT(A.IntT) -> add_terminal builder (L.build_ret (L.const_int
        i32_t 0))
      | _ -> ()

  in

  List.iter build_function_body functions;
  the_module
```

rusty.ml

(* signed: Yanlin Duan, Zhuo Kong *)

(* Top-level of the rusty compiler: scan & parse the input, check the resulting AST, generate LLVM IR, and dump the module *)

type action = Compile

let _ = let action = Compile in let lexbuf = Lexing.from_channel stdin in let ast = Parser.program Scanner.token lexbuf in let ast = Pre_semant.check ast in let sast = Semant.check ast in match action with Compile -> let m = Codegen.translate sast in Llvm_analysis.assert_valid_module m; print_string (Llvm.string_of_llmodule m) “‘